IBB: Improved K-Resource Aware Backfill Balanced Scheduling for HTCondor

Lan Liu, Zhongzhi Luan, Haozhan Wang, Depei Qian

IBB: Improved K-resource Aware Backfill Balanced Scheduling for HTCondor

Lan Liu[1], Zhongzhi Luan[1], Haozhan Wang[1], Depei Qian[1]

[1] Sino-German Joint Software Institute, Beihang University
100191 Beijing, China
{liulan, 07680, wanghaozhan, depeiq}@buaa.edu.cn

Abstract. HTCondor, a batch system characterized by its matchmaking mechanism, schedules job in FCFS way, so its performance is not ideal as expected. Backfilling is a technique to address the above problem. Most backfilling algorithms are based on CPU information and have large room for improvements with considering other resource information. The K-resource aware scheduling algorithm Backfill Balanced (BB) selects backfill job which can best balance the usage of all resources and achieve better performance compared with the classical backfilling algorithm. However, BB does not realize that small jobs' impacts on resource utilization are negligible and they mainly contribute to reduce the average response time. Here we propose the IBB algorithm, which utilizes the characteristics of small jobs to guide a better job selection. We implemented IBB on HTCondor to improve its performance. Experiments results show that IBB can provide up to 60% performance gains in most performance metrics compared with BB.

1 Introduction

HTCondor, a high-throughput distributed system, can produce high job throughput and reliable system performance by using a flexible mechanism ClassAd [1]. However, its dedicated scheduler performs in FCFS which may result in lots of resource fragmentations when the current job is too large. It provides another way called "Best-fit" to skip the current blocked job. However, the blocked job may suffer from starvation when the subsequent jobs are quite small.

An effective way to handle the above problem is backfilling which schedules low-priority small jobs after the current blocked one into execution to utilize the resource fragmentations. There are two typical backfilling methods: Conservative backfilling and EASY backfilling [2]. William Leinberger et al. proposed the K-resource aware scheduling algorithm Backfill Balanced (BB) which can use the additional resource information to select backfill jobs more intelligently than typical backfilling methods [3]. However, BB does not realize that the influence differences on system resource utilization and global system resource status diagram exerted by large jobs and small ones. What's more, it can only select one job to backfill in one traversal.

Our contribution is to provide a variant of backfilling algorithm termed Improved Backfill Balanced (IBB) based on BB. IBB can make use of the additional resource information and the characteristic of small jobs to make more intelligent decisions on

the selection of backfilling jobs. And we implemented IBB on HTCondor which can further improve its job throughput.

This paper is organized as follows: Section 2 describes the related work of past research in parallel job scheduling. Section 3 presents the core thought and application of IBB. Experiments and evaluations are given in Section 4. The conclusion and future work are provided in Section 5.

2 Related Work

As mentioned in Section 1, the system resource utilization and job throughput of HTCondor are not ideal enough. How to utilize the resource fragmentations effectively in parallel job scheduling is one of its most difficult problems.

Backfilling is well known in parallel job scheduling to increase system resource utilization and job throughput. Conservative backfilling can select a small job to backfill from the back of the queue as long as it does not delay the start time of all the jobs in front of it. EASY takes a more aggressive way and selects a job to backfill if it will not delay the start time of the first job in the queue [2]. It is obvious that Conservative backfilling is not as flexible as EASY since it is under the limitation that all front queued jobs cannot be delayed for execution.

Aside from methods mentioned above, a number of variants based on backfilling have been put forward recent years, including Slack-Based backfilling, Relaxed backfilling, etc [4] [5] [6] [7]. Those variants were shown to have lower average slowdown than EASY. However, they may result in worse wait time and higher time complexity. Most important of all, none of them takes any additional job resource requirements and system resource information into consideration, which can easily lead to unbalanced resource utilization and finally harm system performance.

William Leinberger et al. put forward Backfill Balanced (BB) which provides K-resource aware job selection heuristics. BB uses the additional job resource requirements and current system resource status information to guide job selection for backfilling. It can make efficient use of all the available system resource information and experiments show that its performance gains in average response time can up to 50% over classical backfilling scheduling methods.

3 Improved Backfill Balance Scheduling

3.1 Thought of IBB

The classical backfilling algorithms can improve system resource utilization and job throughput by selecting small jobs to execute on resource fragmentations. However, none of them takes any other available resource information into consideration except CPU. The performance degrades since there may be lots of other resource being wasted. BB can achieve better system performance than classical backfilling methods by providing K-resource aware job selection heuristics. BB selects job based on the

overall ability to balance the system resource utilization, its general notion is that if all the resource usages are kept balanced, then more jobs will likely fit into the system, creating a larger backfill candidate pool. BB's evaluation mechanism showed in (1) includes two parts: a balance measure and a fullness modifier. The balance measure showed on the left part of (1) is used to score each backfill job candidate which aims at achieving the best resource utilization balance. The fullness modifier showed on the right is applied to relax the balance criteria as the system gets full by allowing a larger job which achieves a worse balance to be selected over a smaller one which achieves a better balance when the larger one can nearly full the machine. The job with the lowest score indicates a best balance it can up to by the time. The goal of BB is to improve the average response time of the system by improving the response time of small jobs which can best balance the usage of all system resource [3].

$$\frac{\max_i(S_i + R_i^j)}{\sum_{i=1}^{K}(S_i + R_i^j) / K} * \left\{1.0 - \frac{\sum_{i=1}^{K}(S_i + R_i^j)}{K}\right\} \quad (1)$$

The notations are explained in Table 1:

Table 1. Notation explanation

Notation	Explanation
K	The number of resource types
S_i	The current utilization of resource i in the system S
R_i^j	The requirement for resource i by job j

The BB algorithm can significantly improve system performance compared with the classical backfilling algorithms. However, transient imbalance of resource utilization can be neglected, since it is often caused by the execution of a small number of large jobs and will come out of the circumstance once the large jobs terminate. Long-duration imbalance of resource utilization is often caused by successive execution of many large jobs, and it can only be relieved by the execution of large jobs which are complementary in resource requirements as small jobs characterized by short execution time and tiny resource needs can usually do little improvement for the ease of imbalanced resource utilization situation.

As described above, we found that BB doesn't realize the fact that the small jobs' influence on system resource utilization is negligible for their resource requirement is tiny and the execution time is very short relative to large jobs. Another flaw in BB is that in one traversal of the job queue, it can only select one job to backfill, which lowers the efficiency of backfilling and thus affects the average job response time and job throughput.

Here we put forward the Improved Backfill Balanced (IBB) algorithm based on BB. The same with BB, it is K-resource aware as it takes all available resource information into consideration for backfill job selection. Our standpoint is trying to backfill more small jobs as far as possible by taking all available resource information and the characteristics of small jobs into consideration, which, in return, can achieve

considerable performance gains over BB. The difference of IBB from BB mainly lies in the points listed below:

- If the job meets the basic backfilling requirements and will not worsen the resource imbalance in the system, it can be selected.
- In one traversal of the job queue, backfill all the jobs meet the above requirement.

The evaluation mechanism in IBB is shown in (2), if the evaluation result is true, then we select the job for backfilling, skip it if not:

$$\frac{\max_i\left(S_i + R_i^j\right)}{\sum_{i=1}^{K}\left(S_i + R_i^j\right) / K} \leq \frac{\max_i(S_i)}{\sum_{i=1}^{K} S_i / K} \tag{2}$$

The reason for performance improvement which IBB gains over BB lies in the fact that we realized the influence difference on system resource utilization and global system resource status diagram exerted by large jobs and small ones, and try to backfill more small jobs as far as possible to reduce the average response time and improve job throughput more significantly. The algorithm is given in Algorithm 1.

Algorithm 1. Improved Backfill Balanced Algorithm

```
procedure schedule_jobs()
  schedule_head_of_list()
  backfill_jobs()
procedure schedule_head_of_list()
  for all jobs in queue do
    pivot <- first job in the queue
    if  pivot requires <= current idle resource  then
      start job immediately
    else
      break
    end if
  end for
procedure backfill_jobs()
  pivot <- first job in the queue
  t <- time when sufficient resource will be available for pivot
  extra <- extra resource at t not required by pivot job
  cur_avg_util <- currently average resource utilization
  for each job in queue except the first do
    idle_res <- currently idle resource
    if job requires <= idle_res and will finish by t, or job_requires <=
          min(idle_res, extra) then
      avg_util <- average resource utilization with job being backfilled
      if avg_util <= cur_avg_util then
        start job immediately
      end if
    end if
  end for
```

3.3 Apply Backfilling to HTCondor

HTCondor's simple and novel matchmaking algorithm which can produce high system reliability and high job throughput makes it become more and more attractive. However, to ensure simplicity and reliability, its scheduling strategy for parallel jobs just basically follows the FCFS principle and limits the improvement of system performance. Actually backfilling has little impact on system reliability and simplicity since it only needs to traverse the job queue to select jobs for backfilling when the first job in the queue got blocked, with no other complex algorithm and communications needed.

To further improve HTCondor's system performance, we implement IBB on it, the performance gains over the original FCFS and Best-fit scheduling methods are substantial as expected.

4 Experiments and Evaluations

This subsection describes the experiments conducted to evaluate the IBB. As the scale of real environment is limited, we first make performance comparison by simulating to prove the feasibility and effectiveness of IBB on large scale computing systems. As there is little information about jobs' requirements for disk, network, etc, we only consider CPU and memory here. Then we implemented IBB on HTCondor to improve its job throughput, job response time and system resource utilization, which also proves the effectiveness of IBB in modern RJMS.

4.1 Simulation Experiments

We make performance comparison by simulating to prove the feasibility and effectiveness of IBB on large scale computing systems. As the simulator [10] does not provide job throughput and system resource utilization information, we list them as future work and here just provide results for average wait time, average response time and average slowdown. We simulated on the following two kinds of workloads.

- Workload composed of the same job type which we call as Pure Workload.
- Workload composed of mixed type of jobs. We refer it as Mixed Workload.

By simulating on the above two kinds of workloads, we can prove that our algorithm works well in both scenarios, which, to some extent, can generalize all kinds of workloads in real environments.

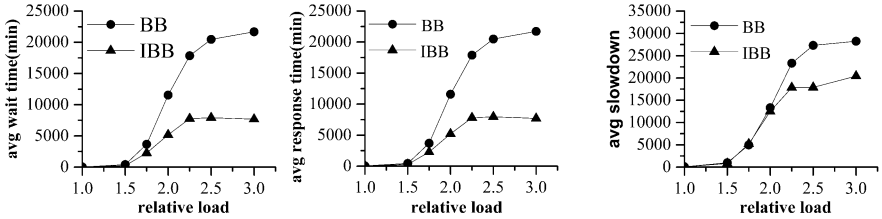
Pure Workload. This workload is provided by the Los Alamos National Lab (LANL) called LANL-CM5. It contains detailed information about job's actual CPU and memory usage, as well as the arrival time and execution time. We simulated the experiments in a 1024-node cluster with 1GB memory per node.

Fig. 1 depicts the performance results for pure workload. The relative load means the experimental system load relative to the workload's original experiment load. The results indicate that IBB is far better than BB. For the average wait time and average response time, the performance gains can up to 65%. The average slowdown has

relative lower gains since it mainly depends on the execution of large jobs. The results show that IBB can work well in environment with a single workload type.

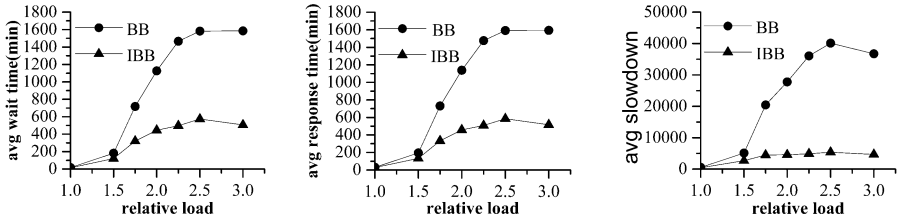
Mixed Workload. To verify that IBB can also work well with mixed types of jobs, we selected two traces to synthetic a mixed workload. The first trace is LANL-CM5 introduced above. The second one called MetaCentrum is memory-intensive and contains several months' worth of accounting records. We processed the synthetic workload in the same way as LANL-CM5.

Fig. 2 depicts the average wait time, average response time and average slowdown for Mixed Workload. The system load changes in the same way as experiment conducted on Pure Workload. The performance gains for the three metrics can up to 60%. Experiment results show that IBB can be applied to mixed job types well and even produces better performance gains than it does on Pure Workload.



a. Average wait time b. Average response time c. Average slowdown

Fig. 1. Average wait time, response time, slowdown under different load with pure workload



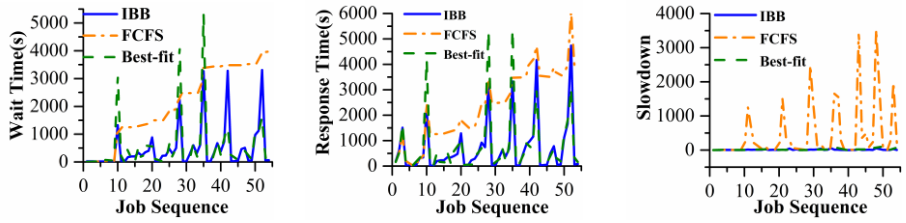
a. Average wait time b. Average response time c. Average slowdown

Fig. 2. Average wait time, response time, slowdown under different load with mixed workload

4.3 Experiments Conducted on HTCondor

Experiment was also conducted on HTCondor to evaluate the performance gains of IBB over its default FCFS and Best-fit scheduling algorithm in real environment. Experiment results prove that IBB can significantly improve HTCondor's performance. The experiment was conducted on a four-node cluster. The job queue was composed of 54 MPI jobs from the NAS Parallel Benchmark and the time interval between each job is one second. Fig. 3 depicts each job's wait time, response

time and slowdown. From Fig. 3(a), (b) we can see that the job parallelism degree scheduled by IBB is much higher than FCFS with quite a number of small jobs being packed to execute on the resource fragmentations while almost no large jobs suffering from starvation. Although the Best-fit scheduling method can get almost the same performance as IBB in job parallelism degree, it can easily make large jobs suffer from starvation, which means it cannot guarantee fairness well. Fig. 3(c) shows that each job's slowdown is almost the same for IBB, which means the wait time of each job endured is negligible compared with its execution time. However, the slowdown is uneven distributed with FCFS scheduling algorithm which indicates worse average job response time.



a. Job wait time b. Job response time c. Job slowdown

Fig. 3. Job wait time, response time, slowdown on HTCondor

4.4 Discussion

To summarize, the experiment conducted on HTCondor are somewhat inconclusive and depend on the job queue. As the scale of our HTCondor pool is limited and the job queue we designed may not strictly conform to the real scenario, the performance gains of IBB in real environment is much better than it got by simulation. To some extent, the experiments conducted on simulator can reflect the true performance of IBB. The success of IBB indicates that taking the characteristic of small jobs and overall system resource information into consideration is reasonable and can select small jobs to backfill in a more efficient way.

5 Conclusion and Future Work

In this paper, we improved the K-resource aware backfilling algorithm by utilizing the characteristics of small jobs and all available resource information, and proposed a new model to select jobs for backfilling more effectively. Meanwhile, we implement backfilling algorithm on HTCondor, which can significantly improve its performance.

The success of IBB shows that small jobs' influence on system resource status diagram is negligible and its contribution to reduce average response time is attractive. IBB can perform a better job packing of the small jobs while maintaining sufficient progress of the large jobs.

While our algorithm performs better than almost all existing backfilling algorithms, it still has some drawbacks. For example, it has to do scheduling based on the job resource requirements information and estimated execution time given by users which may be inaccurate and affect system performance. Future work will focus on putting forward a way to provide appropriate estimation of job resource requirements and execution time to further improve system resource utilization and job throughput.

Acknowledgments. This research is supported by the National Key R&D Program (Grant No. 2016YFB0201403) and the National Natural Science Foundation of China (Grant No. 61361126011, 61133004, 61502019). And here we would like to thank the HTCondor team and the father of HTCondor Miron Livny for providing us help in understanding and using HTCondor.

References

1. Derek Wright, "Cheap cycles from the desktop to the dedicated cluster: combining opportunistic and dedicated scheduling with Condor", Conference on Linux Clusters: The HPC Revolution, June, 2001, Champaign - Urbana, IL.
2. Mu'Alem, A. W., and D. G. Feitelson. "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling." IEEE Transactions on Parallel & Distributed Systems12.6 (2001):529-543.
3. Leinberger, W., G. Karypis, and V. Kumar. "Job Scheduling in the presence of Multiple Resource Requirements." SC Conference IEEE Computer Society, 1999:47-47.
4. Talby, D., and D. G. Feitelson. "Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling." 13th International and 10th Symposium on Parallel and Distributed Processing, 1999.
5. Ward, William A., C. L. Mahood, and J. E. West. "Scheduling Jobs on Parallel Systems Using a Relaxed Backfill Strategy." Job Scheduling Strategies for Parallel Processing. Springer Berlin Heidelberg, 2002:88-102.
6. Nissimov, Avi, and D. G. Feitelson. "Probabilistic Backfilling: Job Scheduling Strategies for Parallel Processing." Springer Berlin Heidelberg, 2007:102-115.
7. Jackson, David, Quinn Snell, and Mark Clement. "Core algorithms of the Maui scheduler." Job Scheduling Strategies for Parallel Processing, 2001.
8. Lawson, Barry G., E. Smirni, and D. Puiu. "Self-adapting backfilling scheduling for parallel systems." International Conference on Parallel Processing 2002:583-592.
9. Michael Litzkow, "Remote Unix – Turning Idle Workstations into Cycle Servers", Proceedings of Usenix Summer Conference, page 381-384, 1987.
10. Frank Cappello, William Kramer, Morris Jette. "CONTRIBUTIONS FOR RESOURCE AND JOB MANAGEMENT IN HIGH PERFORMANCE COMPUTING. " Thesis. 2010.
11. Yi, Shengwei, et al. "Combinational backfilling for parallel job scheduling." International Conference on Education Technology and Computer 2010:V2-112-V2-116.
12. Siyambalapitiya, R., and M. Sandirigama. "New Backfilling Algorithm for Multiprocessor Scheduling with Gang Scheduling." Iup Journal of Computer Sciences (2011).
13. Douglas Thain, Todd Tannenbaum, and Miron Livny, "Distributed Computing in Practice: The Condor Experience" Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.