Malleable-Lab: A Tool for Evaluating Adaptive Online Schedulers on Malleable Jobs

Yangjie Cao*, Hongyang Sun†, Wen-Jing Hsu† and Depei Qian*
*School of Electronic and Information Engineering, Xi'an Jiaotong University, China
Email: caoyj@stu.xjtu.edu.cn, depeiq@xjtu.edu.cn
†School of Computer Engineering, Nanyang Technological University, Singapore
E-mail: {sunh0007, hsu}@ntu.edu.sg

***Abstract*—The emergence of multi-core computers has led to explosive development of parallel applications and hence the need of efficient schedulers for parallel jobs. Adaptive online schedulers have recently been proposed to exploit the multiple processor resource and shown good promise in theory. To verify the effectiveness of these parallel schedulers, it will be reassuring to test them extensively with various parallel workloads. Unfortunately it is still unknown how the job mixes will eventually evolve for multi-core computers; moreover, it is also non-obvious how the parallelism of a typical job will look like. To evaluate the dynamic behaviors of an adaptive scheduler under various scenarios, an ideal workload model for schedulers should thus allow the user to vary parallelism profiles of individual jobs as well as the job arrival patterns. In this paper, we present a tool called Malleable-Lab, which models malleable parallel jobs by extending the traditional moldable job models. Instead of generating a completely random parallelism, which does not allow clear account of the request-allocate responses, we identify several generic patterns of parallelism variations in parallel programs. Using Malleable-Lab we have evaluated two feedback-driven adaptive schedulers, namely, AG-DEQ (Adaptive-Greedy-DEQ) and ABG-DEQ (Adaptive B-Greedy-DEQ), and the well-known scheduler EQUI (Equi-partition). The results reveal that both feedback-driven schedulers outperform EQUI, but on the other hand suffer from high sensitivity to the scheduling overhead. We also found that ABG-DEQ exhibits better transient responses and stability than AG-DEQ. In conclusion, the tool has enabled us to analyze various aspects of the performance of online schedulers, and we have gained valuable insights for adaptive scheduling of parallel jobs on multiple processors.**

I. Introduction

The emergence of multi-core computers has led to explosive development of parallel applications and hence the need of efficient schedulers for parallel jobs. Adaptive online schedulers have recently been proposed to exploit the multiple processor resource [1], [2], [3] and they have shown good promise in theory. To verify the effectiveness of these parallel schedulers, it will be reassuring to test them extensively with various parallel workloads. Unfortunately it is still unknown how the job mixes will eventually evolve for multi-core computers; moreover, it is also non-obvious how the parallelism of a typical job will look like. An ideal parallel workload model for schedulers should thus allow the user to vary the parallelism profiles of individual jobs as well as the job arrival patterns. With the adaptive schedulers, there is a special need to test the responsiveness and stability of a given request-allocate cycle. This need arises because, with adaptive schedulers, the allocation of processors to the jobs is typically done on a periodical basis: the individual jobs request processors based on the need and/or utilization in the past scheduling quantum, and the operating system allocator determines the allotment based on the requests and available resource.

According to the classification by Feitelson, the parallel workload model is generally divided into three types, namely, rigid jobs (which require a fixed number of processors), moldable jobs (where the jobs can run on an arbitrary number of processors at launch time) and malleable jobs (where the jobs can run on an arbitrary number of processors dynamically) [4]. The existing parallel workload models such as Feitelson96 [5], Jann97 [6] and Lublin03 [7] belong to rigid job models and thus lack flexibility in generating jobs with internal parallelism variations. Other parallel workload models are known as moldable models such as Downey97 [8] and Cirne01 [9], which estimate the speedup of a parallel job as a function of its average parallelism and its variance. Downey's model, however, only provides two simple hypothetical parallelism profiles with low and high variance in parallelism and thus lacks information about the internal parallelism characteristics of the parallel applications. Based on the statistical analysis of a survey concerning many users' experiences with parallel machines like IBM SP2, Cirne and Berman [9] provided a more comprehensive moldable job model by taking the partition size of the workload into account. However, this model uses a similar speedup function as Downey's [8], and also does not consider the job's internal parallelism variations. To model the internal job structure, a flexible hierarchical model was proposed by Calzarossa et al. [10] and further extended by Feitelson and Rudolph [4]. At the higher level, a rigid job model is generated from actual workload logs, which provide the external characteristics of parallel jobs such as arrival patterns, work requirements, and average parallelism, etc. Further processing at the lower level adds internal structures to the jobs while maintaining their external properties, such as arrival patterns, work requirements, and average parallelism, etc. Unfortunately, Feitelson provides only two basic internal structures of fork/join and workpile jobs.

In this paper, we present a tool called Malleable-Lab, which models malleable parallel jobs by extending the traditional moldable job models and using a similar hierarchical approach as Feitelson. To represent diverse parallelism variations, we

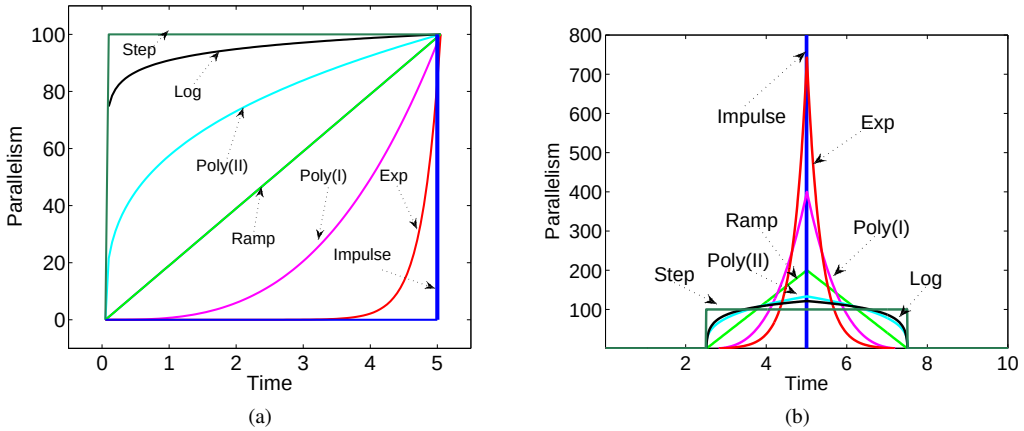

Fig. 1. (a) Seven different types of parallelism variation curves specified by Step, Log, Poly(II), Ramp, Poly(I), Exp, Impulse functions. (b) Seven parallelism variation curves with each one having the same work and length, hence the the same average parallelism.

identify a set of generic internal job structures, which can be mixed flexibly to capture a wide range of running patterns of parallel programs. Using Malleable-Lab, we evaluate two feedback-driven adaptive schedulers that respond to a job's parallelism variations, namely, AG-DEQ (Adaptive-Greedy-DEQ) [1], [11] and ABG-DEQ (Adaptive B-Greedy-DEQ) [3], and the well-known scheduler EQUI [2], [12], which is oblivious to job's parallelism variations. The results reveal that both feedback-driven schedulers achieve better performance than EQUI, but suffer from high sensitivity to the scheduling overhead. Using Malleable-Lab, we also found that ABG-DEQ exhibits excellent transient responses and stability, much better than AG-DEQ. In conclusion, the tool enables us to analyze various aspects of the performance of online schedulers, and we can gain valuable insights for adaptive scheduling of parallel jobs on multiple processors.

II. Malleable Parallel Job Modeling

Many parallel job models exist but very few of them allow generating malleable parallel jobs, which take the internal parallelism variations of the jobs into account. This kind of malleable job model, however, is essential to evaluating adaptive scheduling algorithms whose performance is largely determined by the parallelism variations.

In Malleable-Lab, we present a flexible malleable parallel job model based on traditional moldable job models and use the hierarchical approach, proposed by Calzarossa et al. [10] and extended by Feitelson and Ruldoph [4], for modeling internal job structures. A hierarchical approach helps to reduce the complexity of building parallel workload by decoupling the modeling process into separate levels. At the higher level, jobs are created based on workload logs, which provide external information such as arrival patterns, work requirements, and average parallelism, etc. In this level we integrate existing job models, such as Downey97 [8] and Cirne01 [9], to generate jobs. At the lower level, we propose a framework to construct the malleable jobs by filling in different internal structures which describe the job's parallelism variations over time. It is generally difficult to describe overall parallelism variations of a job, but it is possible to identify segments of the job with specific parallelism structures. In our framework we divide a job into a series of phases and each phase is specified by its internal structure described by several parameters representing the phase's type, degree of parallelism, work requirement, etc. In our model, these parameters are used to control the form of internal parallelism variation and maintain consistency with the external models. Finally, we mix these different internal structures or parallelism variation curves to create a diverse set of parallelism profiles for malleable jobs.

The key task of our framework is how to capture the internal parallelism variations of parallel programs over time. To achieve that, we identify several generic forms of parallelism profiles instead of using completely random parallelism, which can not describe precisely the running patterns of parallel programs. Specifically, we identify seven generic forms of distinct parallelism variation curves, which are specified by Step, Log, Poly(II), Ramp, Poly(I), Exp and Impulse functions, as shown in Figure 1a. These various profiles, which can be further adjusted with different parameters, provide a comprehensive coverage of the parallelism dynamics in different phases: the Step profile describes the more stable parallelism requirement in a given period of time; the Impulse profile represents the drastic variation of parallelism in instant time; the Ramp profile describes linear increasing parallelism; the Exp, Log and two kinds of Poly profiles describe sub-linear and super-linear changing parallelism, respectively. Moreover, these different parallelism variation curves can reflect a wide range of real parallel program running patterns. For instance, the Impulse profile can emulate a drastic one-off increase in parallelism typically encountered in, e.g., a short parallel FOR loop, while the Step profile can represent a more stable data-parallel section of the job. The Ramp profile as well as other profiles can model increases in the job's parallelism with different rates for spawning parallel threads. Figure 2 demonstrates several ideal running parallelism patterns through real parallel program segments for Step, Ramp, Poly(I) and Exp

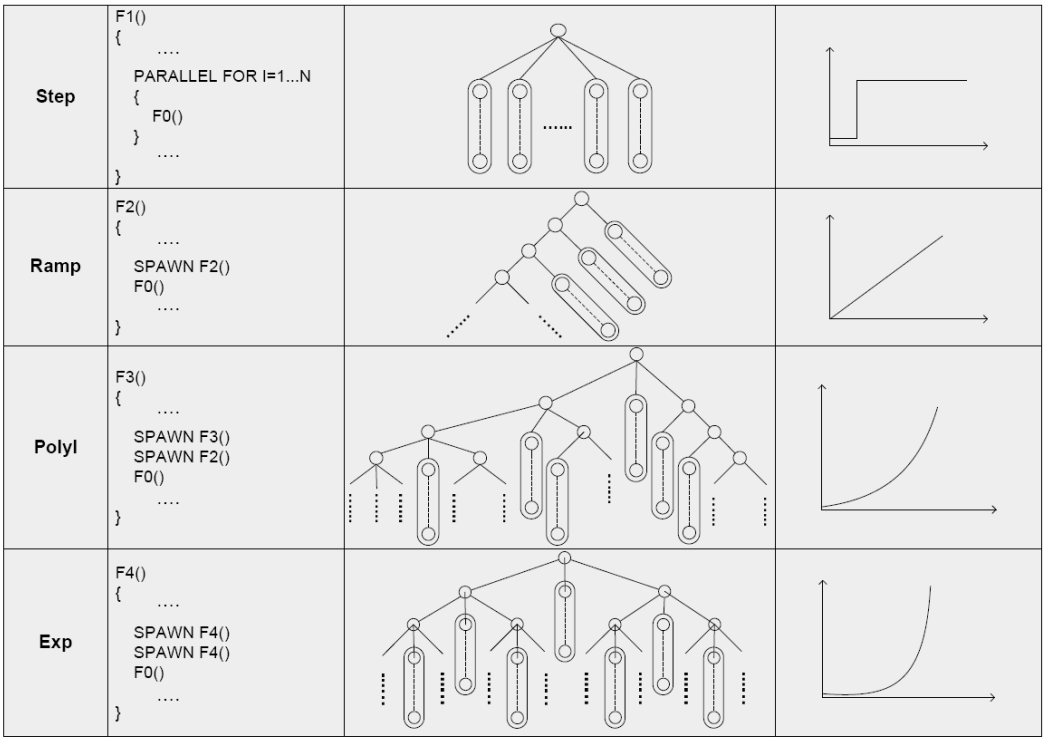

Fig. 2. Sample parallel program segments and their corresponding parallelism variation over time.

profiles. In the figure, function F0() represents a thread with a large amount of computation invoked repeatedly by four different functions constructing the given parallelism profiles. As shown in the figure, function F1() consists of a fully parallelized FOR loop without interdependency profiling the Step curve; functions F2(), F3() and F4() recursively spawns themselves and other threads with different calling patterns, hence creating various rates of increasing parallelism.

These various internal parallelism profiles provide a flexible way to construct malleable jobs whose parallelism changes with time. However it is also a challenge to maintain consistency with the original moldable job model. In MalleableLab implementation, we provide a basic way to maintain such consistency and ensure that all kinds of internal variation curves are coherent with each other. To realize that, we generate the required work, average parallelism and phase length for all parallelism variation curves. Specifically, we combine a pair of increasing and decreasing profiles together to create a basic parallelism variation block, as shown in Figure 1b. We first generate the Step profile which ensures that the work and the average parallelism adhere to those initially generated from the higher level. Secondly, other parallelism variation blocks are derived from the Step profile by varying the degree of internal parallelism curve but with the same phase length and work. Therefore, the aggregation of these different variation curves for a job is ultimately consistent with the original moldable one. We should mention that there also exist other ways, such as keeping the same peak parallelism for each type of variation curve but varying their phase work. These flexibilities allow us to construct malleable workloads with different characteristics which are essential to evaluate the practical performance of adaptive online schedulers.

III. Transient Response of Adaptive Scheduling

Adaptive online schedulers have recently been proposed to exploit the multiple processor resource [1], [2], [3], and they have shown good promise in theory, but they are seldom verified in practice due to the lack of suitable tools. MalleableLab, which provides malleable workload with a comprehensive set of internal parallelism variations, is more suitable for analyzing adaptive schedulers that dynamically allocate processors to jobs at run time. One benefit of this tool is to reveal how fast and how accurately an adaptive scheduler responds to a specific parallelism variation. We refer to such instantaneous reaction to the parallelism variation as *transient response* of the adaptive scheduler, which provides insights on its performance in practice. In this section, we introduce two feedback-driven adaptive scheduling strategies, which are known as AG-DEQ [11] and ABG-DEQ [3]. Both algorithms allocate processors to jobs in a non-clairvoyant manner, since generally an algorithm does not have access to the characteristics of the jobs, such as their work, parallelism, etc. For these feedback-driven adaptive schedulers, the processors are (re)allocated to jobs periodically in scheduling quanta based on each individual job's processor request as well as the operating system's allocation policy. In this section, we briefly describe the two algorithms followed by their transient responses.

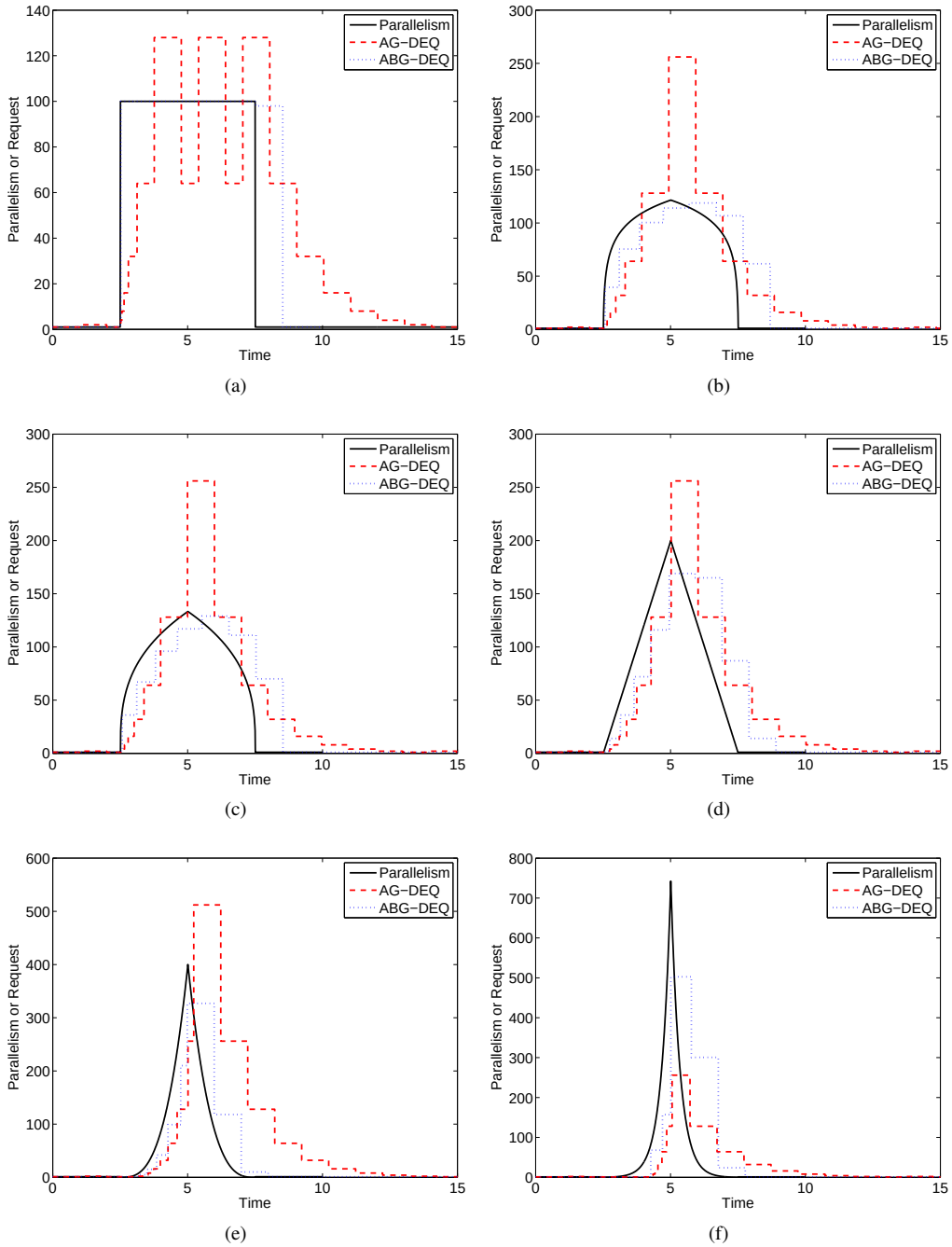

Fig. 3. Transient responses of ABG-DEQ and AG-DEQ with respect to (a) Step, (b) Log, (c) Poly(II), (d) Ramp, (e) Poly(I) and (f) Exp parallelism profile.

A. *Two Adaptive Schedulers*

- AG-DEQ [11]: Each individual job scheduled by AG-DEQ makes a processor request to the operating system for the next scheduling quantum based on its processor utilization in the current quantum. Under ideal allocation, where all requests are satisfied, if the processor utilization in a scheduling quantum reaches a certain threshold, say 80%, the quantum is considered as efficient; otherwise, it is inefficient. If the current quantum is efficient, the processor request for the next quantum will then increase by a factor of 2; it will decrease by a factor of 2 if the current quantum is inefficient. Based on the processor request from each job, the operating system then allocates processors based on DEQ (Dynamic Equi-partitioning) policy [13], which attempts to allocate an equal share of processors to each job, but never allocates more processors to a job than requested. The surplus processors will be given to the other jobs with higher processor requests, if any.

- ABG-DEQ [3]: Unlike AG-DEQ, whose processor requests only respond discretely (by a factor of 2 each time)

with respect to the variations in a job's parallelism, the request of ABG-DEQ attempts to be more representative of the job's immediate parallelism by setting its processor request for the next quantum directly to the job's average parallelism in the current quantum. Upon receiving the processor requests, the operating system uses DEQ policy as well to allocate processors. Note that both AG-DEQ and ABG-DEQ set the initial processor request when the job is first scheduled to 1.

B. Transient Response

We now study the transient responses of the two adaptive schedulers using Malleable-Lab. Figure 3 demonstrates the transient response of AG-DEQ and ABG-DEQ on six generic forms of parallelism profiles (The transient response of the Impulse profile is similar to that of the Step profile and hence is not shown.) In the figure, each profile has the same work, phase length, and average parallelism. The length of the scheduling quantum is set to 1/5 of the phase length, which is scaled in the figures to restore the original parallelism variation. In addition, we add sequential phases before and after each profile such that the processor requests of both schedulers will start and end at a steady state with value of 1. As can be seen in these figures, the two adaptive schedulers exhibit different transient responses with respect to these parallelism variation while their requests are satisfied by the operating system at all time. For the Step profile, AG-DEQ is able to gradually catch up with the parallelism change but suffers from request instability when the parallelism remains constant. In contrast, ABG-DEQ rapidly approaches the parallelism within a quantum, and thereafter provides stable requests by directly utilizing the average parallelism of the job. For the other profiles, both AG-DEQ and ABG-DEQ are able to respond gradually to the parallelism variations with ABG-DEQ in general following more closely the changes of the parallelism and thus taking shorter time to reach steady state. Using Malleable-Lab we found that ABG-DEQ has more effective processor requesting strategy, which suggests that it probably performs better than AG-DEQ in practice. We will verify this claim in the next section through more comprehensive simulations.

IV. Performance Evaluation of Adaptive Scheduling

In this section, we focus on evaluating the performance of these online adaptive schedulers using Malleable-Lab. We develop a simulator by integrating the malleable job model proposed previously to generate the parallel jobs, based on Downey's job model at the high level with the same parameters as set in [8], while we set the system load proportionally to arrival rate of the jobs. We compare two feedback-driven adaptive schedulers AG-DEQ, ABG-DEQ and the well-known scheduler EQUI, which shares the total number of processors equally among all running jobs in the system. We first evaluate the impacts of different parallelism variations on ABG-DEQ, AG-DEQ and EQUI in unconstrained environment where all processor requests of each job are granted. This allows us to evaluate the performances of adaptive schedulers under a favorable circumstance, for otherwise the advantage of a more efficient processor request calculation scheme can not be reflected. Secondly, we simulate a system with 64 processors and generate more realistic workloads, which construct malleable jobs by mixing different internal profiles, to demonstrate the performances of these schedulers. In this case the type of parallelism variation curves of each phase is randomly generated from seven profiles and the average parallelism of each phase is chosen uniformly according to original job model. To compare the performance of different adaptive schedulers, we use the following metrics: average response time, which is the elapsed time from when a job arrives for scheduling to when it completes execution averaged over all jobs, and average utilization, which is the percentage of well-utilized processors averaged over all processors allocated to the jobs.

A. Response Time

The impact of different parallelism variations on response time ratio of ABG-DEQ, AG-DEQ are shown in Figure 4a, which gives the average response time of the three schedulers normalized by the average job length. We do not show response time ratio of EQUI in this figure, since the response time ratio of EQUI is approximately equal to 1 in this unconstrained environment where the processor requests of each job are granted. From the figure we can see that the response time of ABG-DEQ is roughly related to the degree at which the parallelism varies. Specifically, the Step profile contains the more stable parallelism variation and therefore ABG-DEQ has the better performance and the performances become worse with steeper parallelism variations. We should point out that although the Impulse profile has drastic parallelism variation, ABG-DEQ has better response time since it can easily capture its parallelism variation within one quantum and the Impulse profile occurs less frequently. On the other hand, the response time of AG-DEQ is relatively insensitive to different parallelism variations because it is oblivious to the types of internal parallelism variations. Figure 4a clearly shows that ABG-DEQ achieves better response time ratio than AG-DEQ. From the simulation results in this unconstrained environment we can confirm that an adaptive scheduler with better transient responses should be able to achieve superior performances.

Figure 4b shows response time of three adaptive schedulers under a range of workloads, which constructs malleable jobs by mixing different internal profiles. From the figure, we can learn that AG-DEQ and ABG-DEQ significantly outperform EQUI with respect to response time. An obvious reason is that both AG-DEQ and ABG-DEQ take advantage of the parallelism feedback based on the information of execution history while EQUI is oblivious to job's parallelism and thus has bad responsiveness. Only when the system has light workload with a small number of jobs, EQUI shows its advantages because in this case all the jobs can be easily satisfied on the given processors. The simulation results demonstrate that feedback-

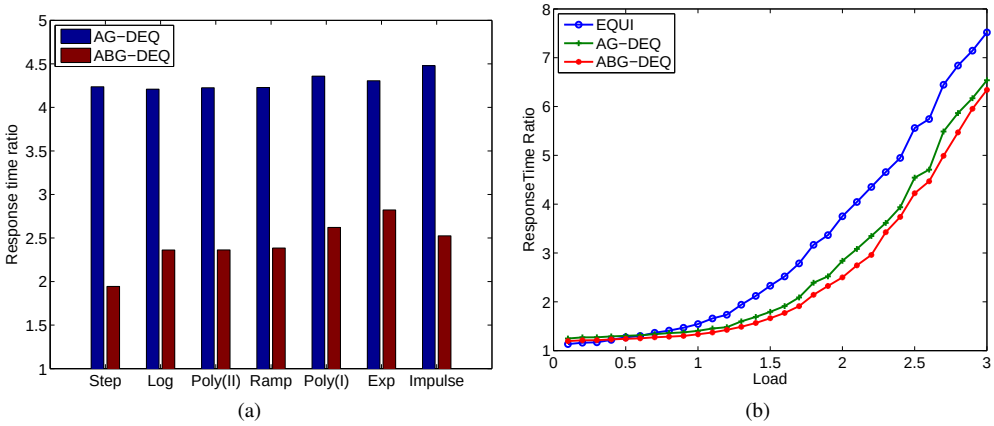

Fig. 4. Response time of ABG-DEQ, AG-DEQ and EQUI on (a) seven different parallelism profiles in ideal, unconstrained environment and (b) a range of workloads with mixed parallelism profiles.

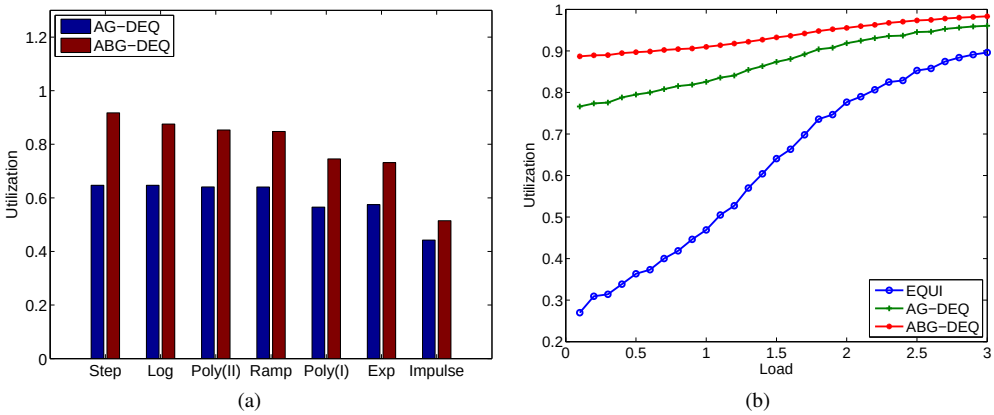

Fig. 5. Utilization of ABG-DEQ, AG-DEQ and EQUI on (a) seven different parallelism profiles in ideal, unconstrained environment and (b) a range of workloads with mixed parallelism profiles.

driven adaptive schedulers are more effective in situations where many parallel jobs with different parallelism variations are competing for limited processor resources. Moreover, Figure 4b also shows that the performance of ABG-DEQ is always better than that of AG-DEQ, which is again due to the more effective processor request feedbacks of ABG-DEQ.

B. Utilization

Figure 5a shows the utilization of ABG-DEQ, AG-DEQ with respect to different internal parallelism variation profiles. Since processors allotted by EQUI are largely wasted in the unconstrained environment, the utilization of EQUI is approximately equal to zero and thus we do not show it in this figure. As can be seen from the simulation results utilizations of two feedback-driven schedulers are significantly impacted by different internal parallelism profiles. In general, smoother parallelism variations tend to give better utilization. From the figure we can clearly see that ABG-DEQ achieves a higher utilization (more than 90%) for the jobs with Step variation and it is also more sensitive to different internal parallelism variations. AG-DEQ, as shown in the figure, is also similar to ABG-DEQ but impacts of parallelism variations are less.

The simulation results conducted for a range of workloads, as shown in Figure 5b, reveal that the utilization of EQUI is significantly influenced by the system load while ABG-DEQ and AG-DEQ are relatively stable. Specifically when the system has light load, where few jobs exist in the system, EQUI has the worst utilization since it is blind to the parallelism and equally allocates all the processor resource to each job which eventually leads to wasting many processor cycles. Only when the system has high load with a large number of jobs, EQUI shows its advantages because in this case all the jobs cannot get enough processor resources although they may have widely changing parallelism. In addition, Figure 5b shows that the utilization of ABG-DEQ is always better than that of AG-DEQ, which is again because of the more effective response to the parallelism variation by ABG-DEQ. The simulation results reconfirm that adaptive schedulers with good transient response to parallelism variation can achieve better utilization.

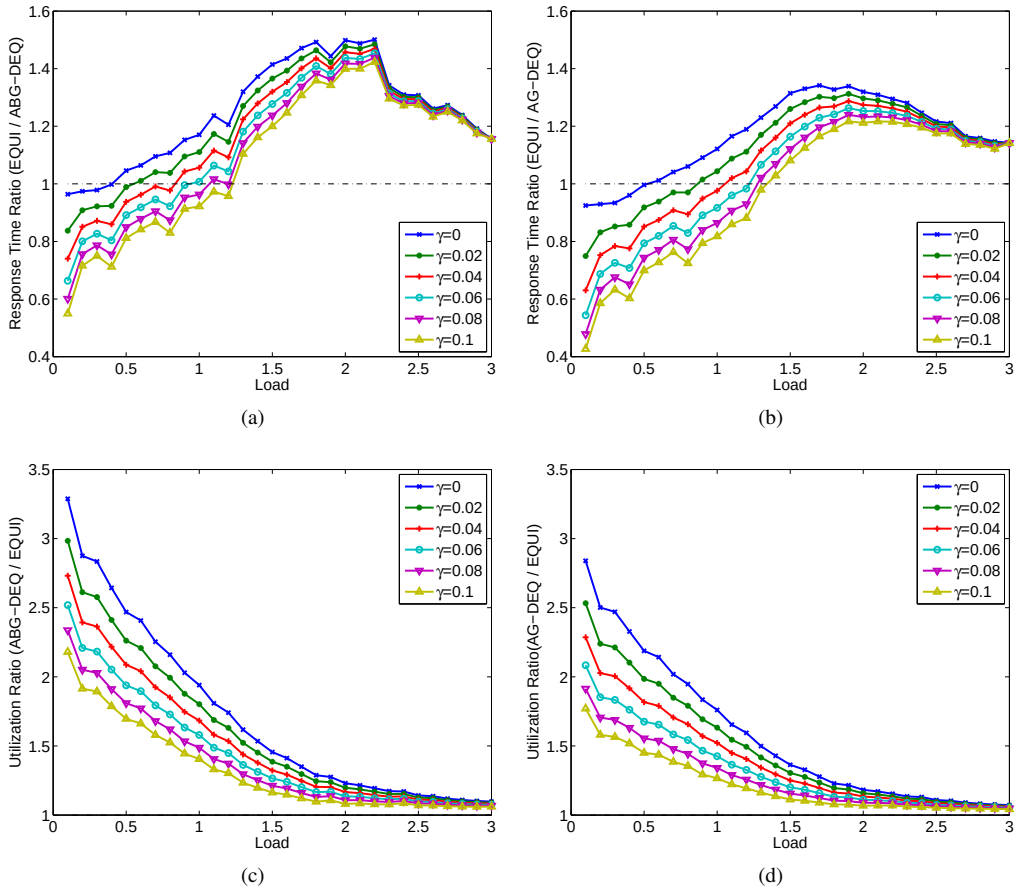

Fig. 6. Impacts of overhead on ABG-DEQ, AG-DEQ and EQUI with respect to response time and utilization

C. *Impacts of overhead on adaptive schedulers*

Using Malleable-Lab, we also conduct a set of simulations to demonstrate the impacts of system overhead on adaptive schedulers. We capture the number of processor reallocations during running time, which are used to measure the reallocation overheads. The response time and utilization of a job under a particular scheduler are then increased by an additive factor $\gamma \cdot \chi$, where χ denotes its number of processor reallocations and γ depends on the system's physical overhead for context switching. Figure 6 shows the response time ratio and utilization ratio of ABG-DEQ and AG-DEQ by comparing them with EQUI. From the figures we can see that although both ABG-DEQ and AG-DEQ achieve better performance than EQUI under most system loads, the two feedback-driven scheduling strategies exhibit degraded performances when the scheduling overhead increases. The reason is that both feedback-driven schedulers adaptively adjust the processor allotment to catch up with internal parallelism variations over time and thus suffer from high sensitivity to the scheduling overhead. On the other hand, the overhead impact on EQUI is less since it only introduces the scheduling overhead when a job finishes or completes. From the simulation results, we can also learn that the response time of ABG-DEQ and AG-DEQ becomes worse than EQUI under light to medium loads, especially when the cost of processor reallocations becomes higher. In contrast, the utilization of ABG-DEQ and AG-DEQ is better than EQUI in this case since feedback-driven schedulers can adaptively adjust the processor requests according to internal parallelism variations. This again demonstrates that the two feedback-driven schedulers are able to provide more effective processor requests. Under heavy system loads, however, the processor requests tend to be deprived and the advantage of feedback strategies diminishes since neither schedulers have direct control over the processor allocations. Therefore, the performances of all adaptive schedulers are similar in such case.

V. Conclusion

In this paper, we present Malleable-Lab for evaluating adaptive online schedulers, which provides a flexible malleable job model based on traditional moldable workload for multiple processors. This malleable job model represents a wide range of running patterns of parallel programs and it is more suitable for evaluating adaptive online scheduling policies whose performance is sensitive to the internal parallelism variations. Using Malleable-Lab we comprehensively studied three adaptive online scheduling strategies and gained valuable insights of them on multi-processor systems. Generally speak-

ing, feedback-driven schedulers work well but suffer from the scheduling overhead. The proposed malleable job model clearly captures the impact of different parallelism variations on the performance of these adaptive schedulers. One of our future work items is to further verify how closely these generic forms of parallelism variations match the running patterns of the real parallel programs. Another interesting direction is to learn how to mix these generic forms, and other possible parallelism variations, to best identify the characteristics of evolving multi-core programs. We believe that such a tool will eventually benefit creation and evaluation of innovative adaptive schedulers.

ACKNOWLEDGMENT

Yangjie Cao and Depei Qian are supported by China National Hi-tech Research and Development Program (863 Project) under the grants No. 2007AA01A127, 2009AA01Z107 and Natural Science Foundation of China under the grant No.60873053. Yangjie Cao worked on this project while he was a visiting scholar at Nanyang Technological University during Summer 2009.

REFERENCES

[1] K. Agrawal, Y. He, W.-J. Hsu, and C. E. Leiserson. Adaptive scheduling with parallelism feedback. In *PPoPP*, pages 100 – 109, New York City, NY, USA, 2006.

[2] J. Edmonds. Scheduling in the dark. In *STOC*, pages 179–188, Atlanta, GA, USA, 1999.

[3] H. Sun, and W.-J. Hsu. Adaptive B-Greedy (ABG): A Simple yet Efficient Scheduling Algorithm. In *SMTPS* in conjunction with *IPDPS*, pages 1–8, Miami, FL, USA, 2008.

[4] D. G. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling Lecture Notes in Computer Science, vol. 1459, pages 185, 1998.

[5] D. G. Feitelson Packing schemes for gang scheduling Lecture Notes in Computer Science, pages 89–111, 1996.

[6] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira and J. Riordan. Modeling of Workload in MPPs. In *JSSPP*, pages 95–116, Geneva, Switzerland, 1997.

[7] U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. Journal of Parallel and Distributed Computing, 2003.

[8] A. B. Downey. A parallel workload model and its implications for processor allocation 6th Intl. Symposium High Performance Distributed Computing, 1997.

[9] W. Cirne and F. Berman. A comprehensive model of the supercomputer workload In 4th Annual Workshop Workload Characterization, pages 140–148, 2001.

[10] M. Calzarossa, A. Merlo, D. Tessera, G. Haring, and G. Kotsis. A hierarchical approach to workload characterization for parallel systems. Lecture Notes in Computer Science, vol. 919, pages 102–109, 1995.

[11] Y. He, W.-J. Hsu, and C. E. Leiserson. Provably efficient two-level adaptive scheduling. In *JSSPP*, pages 1–32, Saint-Malo, France, 2006.

[12] J. Edmonds, D. D. Chinn, T. Brecht, and X. Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. In *STOC*, pages 120–129, El Paso, TX, USA, 1997.

[13] C. McCann, R. Vaswani, and J. Zahorjan. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors ACM Transactions on Computer Systems (TOCS), vol. 11, pages 146-178, 1993.