Enhancing the Functionality of a GridSim-based Scheduler for Effective Use with Large-Scale Scientific Applications

Srishti Srivastava and Ioana Banicescu
Dept. of Computer Science & Engineering
and Center for Computational Sciences
Mississippi State University, Mississippi State, USA
Email: {ss878@, ioana@cse.}msstate.edu

Florina M. Ciorba and Wolfgang E. Nagel
Center for Information Services and
High Performance Computing (ZIH),
Technische Universität Dresden, Germany
Email: {florina.ciorba, wolfgang.nagel}@tu-dresden.de

Abstract

The performance of computationally intensive scientific applications on the underlying system can be maximized by providing application-level load balancing of loop iterates via the use of dynamic loop scheduling (DLS) algorithms. These DLS methods are based on probabilistic analyses, and therefore account for unpredictable variations of algorithmic, systemic and application level characteristics. A considerable number of DLS algorithms has been proposed in the last decade, and some of them have been effectively integrated into scientific and engineering applications, yielding significant performance improvements. However, scheduling scientific applications in large-scale distributed systems where, the chances of failure, such as processor or link failure, are high, makes the problem of achieving a load balanced execution even more challenging. Although real experiments are necessary to verify the benefits of using DLS, they prove to be very time consuming when every level of detail is required for the assessment of the execution of complex, data parallel and irregular scientific applications using DLS on a wide range of architectural platforms and computational environments. Thus, we propose the use of simulators which can give results that are not always experimentally measurable with the current technology. Simulations also help in studying the problem at various levels of abstraction and provide practical feedback.

In this paper, we discuss the implementation of DLS techniques in Alea, a GridSim based scheduling simulator. Based on the simulation results, we further compare the load balancing characteristics of these methods in a simulated parallel and distributed computing environment.

1. Introduction

A parallel and distributed computing system has a set of defined policies for the use of its resources. Most policies include provisions for load balancing, scheduling and fault tolerance. In general, the characteristic of a load balancing and scheduling technique is to adhere to a set of policies of a distributed system. Often these techniques minimize idle time, overloading resources with jobs and control overheads. A number of dynamic loop scheduling (DLS) techniques have been developed to provide load balancing on parallel and distributed systems for the execution of scientific applications with large, computationally intensive loops and irregular loop iteration execution times. Further details regarding the experimental scalability studies of the DLS techniques such as, fixed sized chunking (FSC), guided self scheduling (GSS), factoring (FAC), weighted factoring (WF), adaptive weighted factoring (AWF) and its variants AWF-batched (AWF-B) and AWF-chunked (AWF-C), adaptive factoring (AF) can be found in the literature [12] [13] [2] [4] [8] [14] [6]. In previous studies, these DLS methods have been tested for a number of scientific applications running on ready to use infrastructure for parallel and distributed systems. However, the experiments conducted involved a limited range of problem sizes and number of processors available for running the scientific applications. This also limited the testing of the DLS methods for their scalability and adaptability. The scalability of the DLS techniques has been previously studied experimentally on various architectural and computational environments with limited number of processors. Analyzing the scalability of these DLS methods and scheduling scientific applications on large scale parallel systems requires a framework for modeling and simulation of such systems and the scientific applications. To address and overcome the limitations of testing the DLS methods exhaustively on real systems, we use an event based simulator called Alea [15]. It is a task scheduling simulator built on top of the GridSim simulator [16]. Alea is composed of independent entities which communicate amongst each other through message passing. A detailed description of Alea and the underlying GridSim API is given in Section 2. Over the years modeling and simulation have emerged with the development of a variety of grid schedulers. In this work, we selected Alea as the simulator of choice due to being specifically designed for task scheduling and providing a suitable match for the large number of loop iterates of a scientific application being scheduled on a simulated grid computing system using the DLS methods mentioned earlier. In this work,

978-0-7695-4540-0/11 $26.00 © 2011 IEEE
DOI 10.1109/ISPDC.2011.22

each loop iterate is considered a task to be allocated on a grid resource during the simulation. Being a discrete event-driven simulator, Alea handles events or tasks as they arrive. With the already built-in scheduling techniques, it may not guarantee that events are arriving in good order. For instance, if two different events have the same arrival time, their order in the event queue is unpredictable. These in-built methods do not schedule new jobs until previously scheduled jobs are correctly handled. These techniques do not take into account any unpredictable variations in the system during the execution of the tasks. Thus, to execute the computationally intensive, data parallel and highly irregular scientific applications on a simulated grid computing environment, it was required to implement the DLS methods in Alea. To the best of our knowledge, the DLS methods are implemented for the first time in a simulator in the present work. In contrast to the scheduling techniques already present in Alea, DLS address the variations in the algorithmic and systemic characteristics during application execution. Due to the difference in the scheduling strategies of the DLS methods and the ones in Alea, we had to change the functionality of some of the entities in the simulator and upgrade Alea for compatibility with DLS. The challenges of implementing the DLS techniques in Alea are discussed in detail in Section 3. In this work, we mainly focus on three DLS methods: FSC, GSS, and FAC. The experimental results in Section 4 confirm the performance and the load balancing results that are characteristic to the DLS methods, and the behavior of each of the three techniques. Section 5 discusses the conclusions derived from the simulation experiments, and also gives an overview of the ongoing work which will be a future extension of the work presented in this paper.

2. Background and related work

The advancements in computer technology have led to an increase in the demand of computational power. The motivation behind the use of very large-scale systems and grids of computational resources is to evenly utilize the smallest amount of computational cost. One of the major requirements of these systems is to have maximum load balancing with the help of proper scheduling. A number of scheduling techniques have been developed and exploited over the years to address the issue of load balancing. These techniques have broadly been classified into static and dynamic techniques. The static techniques are helpful in minimizing the individual task's response time and do not have an overhead for information gathering. However, they require prior knowledge of the system and they cannot address unpredictable changes during runtime. On the other hand, the dynamic techniques have been developed to address unpredictable changes, and maximize resource utilization at the cost of information gathering overhead. As we are scheduling complex scientific applications on a grid system architecture, dynamic techniques are most suitable for this scenario for achieving maximum resource utilization, thus capturing the smallest of the available computational cost from a grid resource. The subsections below discuss the DLS methods we use for dynamic scheduling followed by a description of the various grid simulators available for application scheduling. A justification that Alea is the most suitable simulator for implementing the DLS methods is also given in the following sections.

2.1. Dynamic loop scheduling algorithms

A number of DLS algorithms have been developed in the recent years to schedule scientific applications on parallel and distributed environments in a load balanced manner. The studies done in [12], [13], [2], [4], [8], [14], and [6], evaluate the performance of the various DLS methods analytically and experimentally. However, these studies did not include a wide range scalability comparison experimentally, as the large systems are not easily available in practice. The DLS algorithms have been developed to schedule scientific applications which often run for several time steps and consist of large number of loops with several loop iterations. These applications run on a real time parallel and distributed system and their execution may exhibit unpredictable variations due to algorithmic and systemic characteristics. The larger the size of the application and the underlying computational system gets, the more challenging the problem gets for the DLS methods. Although, the experiments with real world applications and real machines show that the DLS methods are adaptable to any changes because they are based on probabilistic analyses, these experiments are not able to demonstrate the scalability of these techniques on large scale systems due to the limitation of fewer resources in the real world small testbed sizes. The initial techniques, FSC and GSS, have been evaluated for a very large number of processors analytically. There are no experimental results for these DLS methods on such large scale real systems.
The scalability analysis of these DLS methods was obtained from the experiments conducted in the previous studies on a limited number of resources, and to study the DLS scalability on large number of processors there is a need for implementing the DLS methods using an application scheduling grid simulator. To the best of our knowledge, this is the first time that these DLS methods (FSC, GSS, and FAC) are implemented in a simulator. A discussion on the various types of grid simulators available to be used for job or task scheduling, and their comparison with the most suitable simulator for task scheduling, Alea, is presented in the following subsection.

2.2. Grid simulators and Alea

The recent development in modeling and simulation of real world systems has led to the development of

many standard and application specific tools and technologies. These also include standard simulation libraries such as SimJava [17]. However, despite the increase in the development of such simulation tools, there are very few application scheduling simulators available for a grid computing environment. Among these, the most popular simulators are: Bricks [18], MicroGrid [19], SimGrid [20], and GridSim [16]. Bricks [18] was developed at the Tokyo Institute of Technology in Japan. This simulator has only been designed for a centralized global scheduling methodology as opposed to the DLS methods which can be implemented both in central or distributed scheduling methodology. MicroGrid [19] is an emulator developed at UCSD. It provides precise emulation results; however, the process of modeling scheduling scenarios for realistic analysis of the scheduling techniques comes with a large development overhead. SimGrid [20] is a C based simulation toolkit for application scheduling developed at UCSD. This is one of the most powerful and efficient, real time simulators. However, to the best of our knowledge, it is used to simulate single scheduling entities and time-shared systems.

The scientific applications scheduled using the DLS methods come with multiple jobs or tasks competing for the same resource, and each of these tasks come with their own irregular execution times. This requires the simulator to have support for space shared resources as well. GridSim [16] has been developed under the Gridbus project at GRIDS Lab at the University of Melbourne. It extends the basic model provided by the simulators discussed above (i.e., SimJava) and overcomes their limitations accordingly. It provides a wide range of heterogeneous resources with different configurations. Alea [15] is a task scheduling simulator built on top of the GridSim 5.0 simulator API, and is well suited for the implementation of our DLS methods. Alea extracts the required API from the GridSim toolkit, which are related specifically to task scheduling. This makes the use of Alea easier for simulating the execution of the loop iterates of a scientific application on a grid environment through the DLS methods. Alea is comprised of independent modules, such as, a centralized scheduler, a job generator, a job submission system, and a resource manager. These modules communicate with each other using SimJava's communication infrastructure, wherein each simulation entities runs in its own thread, connects to other entities by ports and communicates with other entities by sending and receiving event objects. These are also the basic modules which we modified to simulate a scheduling environment for the DLS methods, the scientific applications and the grid resources. The incorporation of the DLS methods in Alea is discussed in more detail in the following section.

3. Integrating DLS within Alea

The basic structure of the Alea simulator, used for the simulation of scheduling loop iterates modeled as individual tasks of a scientific application on a grid environment, is illustrated in Figure 1, which is a modified version of the respective figure illustrated in [26]. The original functionality of the Alea framework is described in the [15] and [26].

The first step for running the experiments is to generate

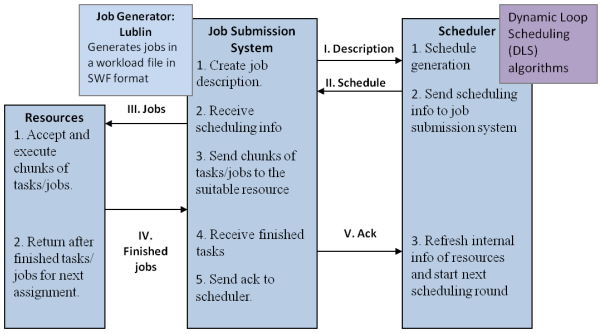

Figure 1. **Architecture of Alea's functionality extended with DLS algorithms**

the application tasks (jobs) and their characteristics. We used a workload generator called Lublin [21] for generating a workload file in the standard workload format (SWF) [22]. The workload file carries the `.swf` extension and consists of the specified number of jobs or tasks of the application and their characteristics. The execution times of these tasks follows a hyper-gamma distribution to simulate the irregular loop iteration execution times within a scientific application. The hyper-gamma distribution has already been implemented in Lublin [21] to generate the running times of various tasks. This workload file is then read by the job submission system to create job descriptions in the form of gridlets and to send these gridlets to the scheduler entity.

The grid resource entity reads the number and characteristics of the resources from a machine file which is generated offline along with the workload file. The machine file carries the extension `.swf.machines` and contains a list of all the grid resources and their respective configurations. Alea allows a resource to be configured with several machines, and each machine to be configured with several processors. As Alea only allows mapping of one task to one processor at a time, it was a challenging task to implement the DLS methods by which a chunk of tasks is assigned to a processor at a time. To overcome this problem, we configured each grid resource to consist of a single processor by declaring in the machine file only one machine and a single processor to a grid resource. This enabled assigning more than one task to a resource or, in our case, a processor.

The implementation of the three DLS methods which are, FSC, GSS, and FAC, in Alea requires their incorporation into the scheduler module, i.e., the `Scheduler.java` class. This module originally comes with a set of predefined scheduling techniques for allo-

cating incoming jobs onto grid resources. This is the part where we implement the DLS methods so that the tasks simulating the behavior of the loop iterations of a scientific application are scheduled on the required resources using the appropriate DLS methodology. The main challenge of implementing the DLS techniques was to change the property in Alea which schedules jobs as they arrive in contrast to the assumptions considered by the DLS methods. This property is useful when the total number of jobs is unknown, as it is the case of a grid scheduler. However, when scheduling scientific applications, all the application tasks must be acquired in a queue before the DLS method schedules them on different resources based on various DLS policies. This is justified since for any scientific application we would know the total number of loop iterates in advance. Another challenging implementation problem was to hold the execution of gridlets onto a resource until a chunk of tasks (or gridlets) has been assigned to it during a scheduling step. However, in the other techniques implemented in Alea, a job would be mapped to a suitable resource, or a processor as soon as it arrived, and would be sent for execution without waiting for the arrival of the next job in the queue. After the scheduler assigns and executes the application tasks onto the resources according to the scheduling policies provided by the DLS methods, it sends the execution results, such as the makespan value, the resource utilization, the scheduling overhead, and others, to the results collector entity, which in turn, reports these statistics to the user. The results are then displayed on the screen of the computer on which the simulation is conducted, and are also saved in the output files created by Alea at the end of the experiment. Alea uses the visual characteristics of the Java libraries and the NetBeans environment (which is an interactive development environment or IDE for Java developers) to create graphical results which enhance the analysis of the experimental results.

An analysis and comparison of the performance of the DLS methods is presented in the following section. The performance evaluation of the DLS methods is based on the values obtained from the simulation results generated with Alea and plotted in the following charts.

4. Experimental results

The complexity of the simulation experiments depends upon the number of resources, number of tasks and the environmental characteristics. We used several sizes of testbeds for running the experiments with the three DLS methods. We created resource files with up to 2^{10} processors. Several workload files were created with up to 2^{18} tasks. The experiments were conducted for scheduling the tasks in each workload file on all the resource files, one by one, using one of the three DLS methods at a time.

All simulation experiments were performed on a 6GB SUSE Linux machine with eight Intel(R) Xeon(R) 3.2GHz

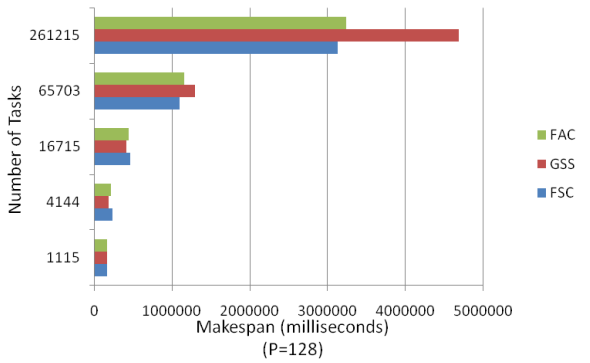

Figure 2. **Makespan obtained from simulated execution of different number of tasks on 128 resources using FSC, GSS, and FAC**

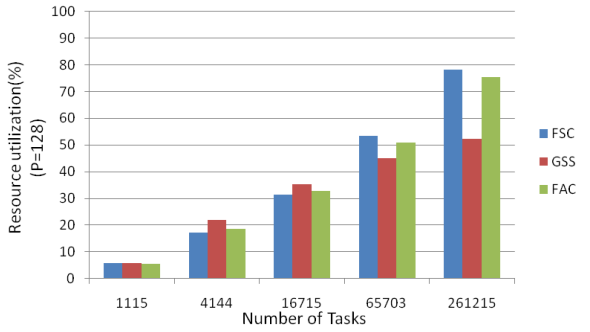

Figure 3. **Percentage of resource utilization obtained from simulated execution of different number of tasks on 128 resources using FSC, GSS, and FAC**

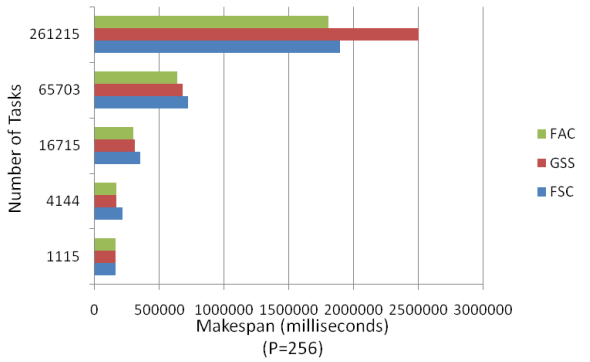

Figure 4. **Makespan obtained from simulated execution of different number of tasks on 256 resources using FSC, GSS, and FAC**

processors. The DLS techniques were implemented using the Java programming language and incorporated into Alea [15]. The simulation experiments were run in the NetBeans environment. The heap memory size allocated to the Java Virtual Machine was 3GB for running the experiments to simulate very large clusters, ranging from

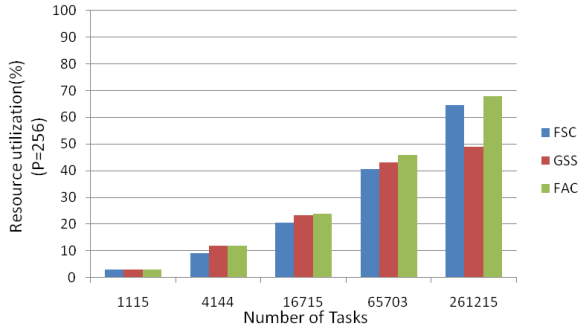

Figure 5. **Percentage of resource utilization obtained from simulated execution of different number of tasks on 256 resources using FSC, GSS, and FAC**

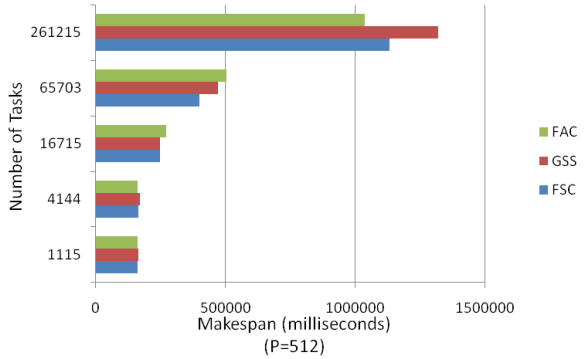

Figure 6. **Makespan obtained from simulated execution of different number of tasks on 512 resources using FSC, GSS, and FAC**

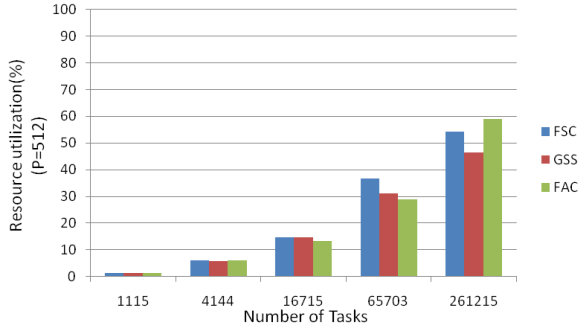

Figure 7. **Percentage of resource utilization obtained from simulated execution of different number of tasks on 512 resources using FSC, GSS, and FAC**

2^4-2^{10} processors, and to simulate scientific applications with very large number of loop iterates, ranging from approximately 2^4-2^{18} iterates.

Simulating the loop iterates of a scientific application: A workload file is created by the workload generator, Lublin [21], which contains a number of tasks with their properties in the tab-delimited SWF [22] format. These tasks synthesize the loop iterates of a scientific application.

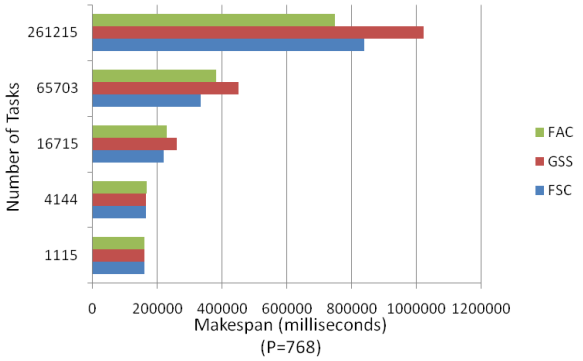

Figure 8. **Makespan obtained from simulated execution of different number of tasks on 768 resources using FSC, GSS, and FAC**

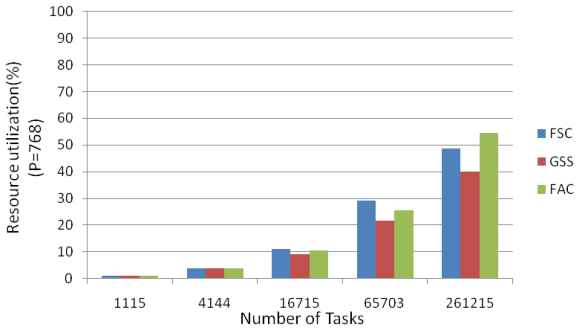

Figure 9. **Percentage of resource utilization obtained from simulated execution of different number of tasks on 768 resources using FSC, GSS, and FAC**

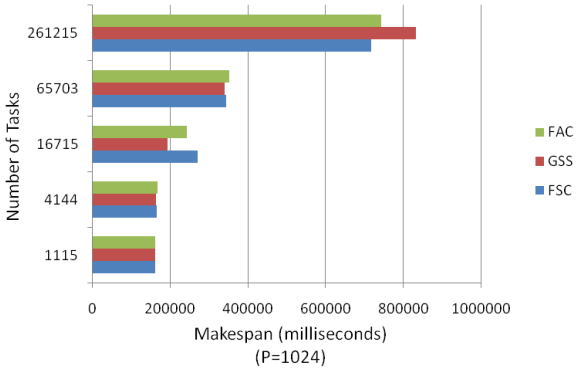

Figure 10. **Makespan obtained from simulated execution of different number of tasks on 1024 resources using FSC, GSS, and FAC**

The properties of these tasks include the start time of the task which, in our case is the same for all tasks, as the total number of iterates is known in advance, the number of processors required by individual tasks, which, in our case is 1 for each task, the running time of the task and, the type of the task, such as, batch tasks (type 21),

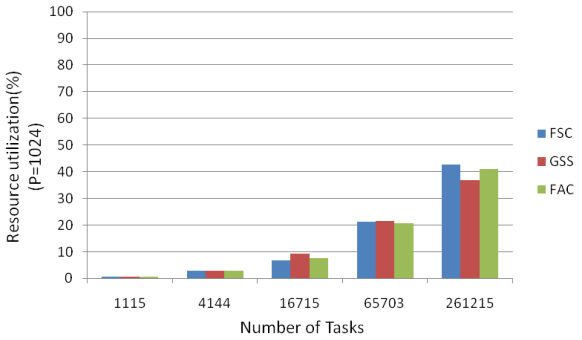

Figure 11. **Percentage of resource utilization obtained from simulated execution of different number of tasks on 1024 resources using FSC, GSS, and FAC**

interactive tasks (type 22), and others. In this work, the workload files consisted of batch tasks of type 21. The running times of the various tasks are generated using a hyper-gamma distribution, which is already implemented in Lublin [21], to simulate the irregular iteration execution times of the loop iterates in an irregular scientific application. For the hyper-gamma distribution, if hg = hyper-gamma(a_1,b_1,a_2,b_2,p) then the expectation and variation are: E(hg) = p·a_1·b_1 + (1-p)·a_2·b_2 and Var(hg) = p^2·a_1·b_1^2 + $(1-p)^2$·a_2·b_2^2 respectively. In this experiment the values of a_1, b_1, a_2, b_2 and, p are $1.2, 0.5, 7, 0.7$ and, 0 respectively.

Simulating the grid cluster environment: A machine file is created to simulate a computing system as a cluster of grid resources. The file contains a list of resources and their properties in a tab-delimited format. The properties of the resources include the resource ID, label of the cluster that this resource belongs to, the number of machines in each resource, and the CPU rating (in MIPS) of each machine. In this work, one resource has been configured to contain a single processor. Thus, each resource has a single machine, each machine has a single CPU and each CPU has the same rating. Thus, the simulated environment is homogeneous, consisting of nodes with a single-CPU and equal speeds.

Implementing the DLS techniques into the scheduler entity: The scheduling methods are implemented in a separate Java class, called `Scheduler.java`. This class is responsible for the behavior of the scheduler entity. Alea already contains a set of queue-based or schedule-based scheduling algorithms implemented in this file. In this work, the three DLS methods, FSC, GSS and, FAC, are also implemented in `Scheduler.java` as queue-based policies, such that the scheduler entity simulates the behavior of these techniques for scheduling the tasks on the grid resources.

Analysis of results: In this work, an analysis of the makespan value and the resource utilization has been illustrated to evaluate the performance and scalability of the three DLS methods. Figures 2, 4, 6, 8, and 10 illustrate the makespan (measured in milliseconds) obtained after the different number of tasks are scheduled and executed on 128, 256, 512, 768 and 1024 resources, respectively, using each of the three DLS methods. The DLS methods strive to minimize the makespan via load balancing for achieving an optimal performance. Among the three DLS methods, FAC gives a minimum makespan in most of the cases. It can be seen that for larger number of tasks and for larger number of processors, FAC is a better choice for minimizing the makespan. However, for smaller number of tasks and smaller number of resources, all three DLS methods yield comparable results due to the fact that there is not much load imbalance in the system. Similarly, Figures 3, 5, 7, 9, and 11 show the resource utilization for the different number of tasks when scheduled and executed on 128, 256, 512, 768 and 1024 resources, respectively, using the three DLS methods individually. For a larger number of tasks, FAC provides a load balanced and optimal resource utilization. For smaller number of tasks, in some of the test cases the three DLS methods have a similar performance, while in other cases FSC or GSS perform slightly better than FAC. These results are in confirmation with the analytical and experimental results obtained for these DLS methods in previous studies [12] [13] [2] [4] [8] [14] [6].

5. Conclusions and future work

In previous studies [12] [13] [2] [4] [8] [14] [6], performance evaluation of the DLS methods has been obtained either analytically, or experimentally, using real world applications executing on real physical machines. However, the results obtained from these real experiments were limited in addressing the scalability and the adaptability of the DLS methods on very large-scale grid clusters. To overcome this limitation, the work presented in this paper, employs simulation of scheduling applications on a cluster of grid resources using the DLS techniques. This paper focuses on evaluating the performance of three DLS methods, FSC, GSS, and FAC. The simple framework provided by Alea helped in efficient implementation and integration of the DLS methods. Using Alea for scheduling the scientific applications on a cluster of grid resources using the three DLS methods, we were able to simulate the execution of different types of irregular loop iteration execution times, on different sizes of computing systems with the required level of detail to assess the performance, scalability, and the load balancing characteristics of these methods.

The immediate ongoing and future work plans include: (i) implementation of other DLS methods, such as, WF, AWF, AWF-B,C and, AF, in Alea to assess their characteristics and validate the simulation results with the real experimental results; (ii) implementation of system load variation and resource failure mechanisms in Alea to test the level of adaptivity of all DLS methods; and (iii) extension of the analytical studies performed in [23], [24],

and [25], for calculating the robustness of DLS methods with simulation-based experimental studies that will lead to an evaluation of their flexibility and resilience against load variations and processor failures, respectively.

Acknowledgment

The authors would like to acknowledge the support of Dalibor Klusáček, the developer of Alea for his support regarding Alea's functionality, leading to the current work. The authors would also like to thank the National Science Foundation for its support of this work through the following grant: NSF #1034897.

References

[1] T. Kunz: The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme. IEEE Trans. on Soft. Eng., 725–730 (1991)

[2] S. Flynn-Hummel, E. Schonberg, and L.E. Flynn: Factoring: A Method for Scheduling Parallel Loops. Comm. of the ACM. 35:8, 90–101 (1992)

[3] I. Banicescu and S. Flynn-Hummel: Balancing processor loads and exploiting data locality in n-body simulations. Procs. of Supercomputing 95 (1995)

[4] S. Flynn-Hummel, J. Schmidt, R.N. Uma, and J. Wein: Load-Sharing in Heterogeneous Systems via Weighted Factoring. Procs. SPAA, 318–328 (1996)

[5] A. Hurson, J. Lim, K. Kavi, and B. Lee: Parallelization of DOALL and DOACROSS Loops: A Survey. Advances in Computers, 45 (1997)

[6] I. Banicescu and V. Velusamy: Load Balancing Highly Irregular Computations with the Adaptive Factoring, Procs. IPDPS '02, 195 (2002)

[7] I. Banicescu and V. Velusamy: Performance of Scheduling Scientific Applications with Adaptive Weighted Factoring. Procs. IPDPS '01, 84 (2001)

[8] I. Banicescu, V. Velusamy, and J. Devaprasad: On the Scalability of Dynamic Scheduling Scientific Applications with Adaptive Weighted Factoring. Journal of Cluster Computing, 6:3, 215–226 (2003)

[9] R. L. Cariño, I. Banicescu, T. Rauber and G. Ruenger: Dynamic Loop Scheduling with Processor Groups. Procs. Int'l Conf. P.&D. Comp. Systems (PDCS 2004), 78–84 (2004)

[10] R. L. Cariño and I. Banicescu: A Framework for Statistical Analysis of Datasets on Heterogeneous Clusters. Int'l Conf. on Cluster Comp., 1–9 (2005)

[11] R. L. Cariño and I. Banicescu: A Dynamic Load Balancing Tool for One and Two Dimensional Parallel Loops. 5th Int'l Symp. on Parallel and Distributed Computing (ISPDC '06), 107–114 (2006)

[12] C. P. Kruskal and A. Weiss: Allocating Independent Subtasks on Parallel Processors. Software Engineering, IEEE Transactions on, SE-11:10, 1001–1016 (1985)

[13] C. D. Polychronopoulos and D. J. Kuck.: Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers. Computers, IEEE Transactions on, C-36:12, 1425–1439 (1987)

[14] R. L. Cariño and I. Banicescu: Dynamic Load Balancing with Adaptive Factoring Methods in Scientific Applications. J. Supercomput. 44:1, 41–63 (2008)

[15] D. Klusáček and H. Rudová: Alea 2: Job Scheduling Simulator. Procs. 3rd Int'l ICST Conf. on Simulation Tools and Techniques (SIMUTools 2010)

[16] R. Buyya and M. Murshed: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. The Journal of Concurrency and Computation: Practice and Experience (2002)

[17] F. Howell and R. McNab: Simjava: A Discrete Event Simulation Library for Java. International Conference on WebBased Modeling and Simulation 30, 51–56 (1998)

[18] K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, S. Sekiguchi, and U. Nagashima: Performance Evaluation Model for Scheduling in a Global Computing System. The International Journal of High Performance Computing Applications, 14:3, 268–279 (2000)

[19] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien: The MicroGrid: A Scientific Tool for Modeling Computational Grids. Proc. IEEE Supercomputing (SC 2000), 127–141 (2000)

[20] H. Casanova: Simgrid: A Toolkit for the Simulation of Application Scheduling. Cluster Computing and the Grid, Proceedings. First IEEE/ACM International Symposium on, 430–437 (2001)

[21] U. Lublin and D.G. Feitelson: The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs. J. Par. and Dist. Comp., 63:11, 1105–1122 (2003)

[22] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby: Benchmarks and Standards for the Evaluation of Parallel Job Schedulers. In IPPS/SPDP 99/JSSPP 99: Proc. of the Job Sch. Strat. for Par. Processing, 67–90 (1999)

[23] I. Banicescu, F. M. Ciorba, and R. L. Cariño: Towards the Robustness of Dynamic Loop Scheduling on Large-Scale Heterogeneous Distributed Systems. Parallel and Distributed Computing, International Symposium on, 129–132 (2009)

[24] S. Srivastava, I. Banicescu, and F. M. Ciorba: Investigating the Robustness of Adaptive Dynamic Loop Scheduling on Heterogeneous Computing Systems. Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW), IEEE International Symposium on, 1–8 (2010)

[25] S. Srivastava, F. M. Ciorba, and I. Banicescu: Employing a Study of the Deterministic Robustness Metrics to Assess the Reliability of Dynamic Loop Scheduling. Proceedings of the IEEE High Performance Computing Conference (HiPC 2010) - Student Research Symposium (2010)

[26] D. Klusáček, L. Matyska, and H. Rudová: Alea: Grid Scheduling Simulation Environment. Proceedings of the 7th International Conference on Parallel Processing and Applied Mathematics (PPAM'07), 1029–1038, Berlin, Heidelberg (2007)