Towards the robustness of dynamic loop scheduling on large-scale heterogeneous distributed systems

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

Florina M. Ciorba and Ricolindo L. Cariño
Center for Advanced Vehicular Systems
Mississippi State University
Mississippi State, USA
Email: {florina,rlc}@cavs.msstate.edu

Abstract

Dynamic loop scheduling (DLS) algorithms provide application-level load balancing of loop iterates, with the goal of maximizing application performance on the underlying system. These methods use run-time information regarding the performance of the application's execution (for which irregularities change over time). Many DLS methods are based on probabilistic analyses, and therefore account for unpredictable variations of application and system related parameters. Scheduling scientific and engineering applications in large-scale distributed systems (possibly shared with other users) makes the problem of DLS even more challenging. Moreover, the chances of failure, such as processor or link failure, are high in such large-scale systems. In this paper, we employ the hierarchical approach for three DLS methods, and propose metrics for quantifying their robustness with respect to variations of two parameters (load and processor failures), for scheduling irregular applications in large-scale heterogeneous distributed systems.

1. Introduction

Researchers and scientists from various fields are interested in the accurate modeling and simulation of various complex phenomena from various scientific areas. These simulations are often routines that perform repetitive computations (in the form of DO/FOR loops) over very large data sets, and the number of repetitive computations (iterations) in these codes is not always constant. Moreover, their nature (or computational requirements) may be irregular, making one iteration likely to take more time than others, depending on the simulation. The resources in a large-scale system are widely distributed and highly heterogeneous, and as such, are usually shared among multiple users, and their availability cannot always be guaranteed or predicted. Hence, the *quality* and *quantity* of resources available to a single user changes continuously.

In this work, dynamic loop scheduling (DLS) techniques are considered to be the key solution for achieving and preserving the best performance of these applications in such environments. Herein, it is considered that a '*loop iteration*' (or a chunk of loop iterations) with variable execution time refers to a '*task*' (or a chunk of tasks, among many others within a loop of tasks) with variable execution time. A comprehensive description has been given earlier in a survey by Hurson et al. in [5] and in the relevant literature after that.

DLS methods provide two alternative approaches, *non-adaptive* and *adaptive*, for achieving good load balancing on variably loaded resources, as well as for executing tasks with varying execution times. Most of the techniques described in [5] are based on probabilistic analyses and are non-adaptive. Other non-adaptive techniques, which were not mentioned in the survey above, include fractiling [3] and weighted factoring [4]. Subsequent efforts gave birth to more elaborate techniques, called adaptive, and a few examples are given in [6][8][12]. Most of the above adaptive methods are based on probabilistic analyses, and use a combination of runtime information about the application and the system, in order to predict the system capabilities for the next computational assignments, or to estimate the time future tasks will require to finish execution, in order to achieve the best allocation possible for optimizing application performance via load balancing. In this paper, we employ a hierarchical management approach, and concentrate on two non-adaptive techniques, factoring [2] (FAC) and weighted factoring [4] (WF), and one adaptive technique, adaptive weighted factoring [7] (AWF). These techniques use probabilistic analyses to dynamically compute the size of chunks (a collection of tasks) at run-time, such that they are executed before their optimal time with high probability. Due to space limitations, the interested reader is referred to the appropriate references for details of the above DLS algorithms.

The performance of DLS methods using hierarchical management has been shown to be better than that of the centralized management approach [9][10][11]. Figure 1 illustrates the centralized management approach (left), and the distributed management approach (right). The coordination of and interactions between the processors in the first case are straightforward, whereas for details of the hierarchical management approach, due to space limitations, the interested reader if referred to [9].

Motivation Scheduling applications on large-scale platforms, where chances of faults are high, require an approach based on hierarchical management and mechanisms to ensure the robustness of the DLS methods. For this reason we consider FAC, WF and AWF, which are inherently robust because their design enables them to address unpredictabilities

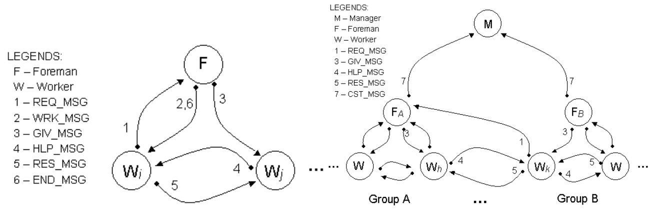

Figure 1. Left: centralized management, right: hierarchical management system

in the application and the system. Previously, the robustness of resource allocations/task scheduling algorithms was addressed individually for a single method, or even for a single application.

Contribution Inspired by the results in [9] we propose to employ the hierarchical approach in the DLS methods above. As such, the processors are organized into dynamically selected groups, such that the physical structure of the underlying platform is easily and well captured. The main contribution of our work lies in using the methodology proposed by Ali et al. [13] to propose two *metrics* that quantify the robustness of the hierarchical FAC, WF and AWF DLS algorithms, used for scheduling a very important *class* of applications: *irregular* tasks, against variations of two system related parameters: *load* and *processor failures*. The proposed metrics in conjunction with the hierarchical DLS methods provide quantitative and qualitative information such as: level of performance, quality of execution, for irregular applications in uncertain large-scale heterogeneous systems. These metrics alone are not more useful or better than any other performance measurement metrics (e.g. makespan, communication cost, resource utilization). Therefore, their use is mandatory towards achieving robustness for hierarchical DLS algorithms running on such uncertain systems.

The paper is organized as follows. Section 2 shows how to design robustness metrics and describes the two proposed metrics. Details regarding the implementation of these metrics and their usefulness are outlined in section 3. The paper is concluded in section 4 with an outline of directions for future work.

2. Robustness metrics design

Robustness is an emergent multifaceted phenomenon in advanced computing systems.

Designing specifically for robustness, to its full extent, is however not yet possible. Robustness benchmarks (or metrics) can be useful to measure *how* a scheduling method reacts to possible erroneous inputs or environmental factors, and hence, should employ mechanisms for *detecting* and *identifying* any erroneous parameters. In this work, we use the FePIA (features-perturbation-impact-analysis) procedure (see [13] and references therein for details), to design metrics that *model* and *estimate* the robustness of hierarchical DLS algorithms on large-scale realistic platforms against two perturbation parameters. The FePIA procedure consists of four general steps:

S.1 *Identify the performance features.*

S.2 *Identify the perturbation parameters.*

S.3 *Identify & clarify the impact of perturbation parameters* (S.2) *on performance features* (S.1).

S.4 *Identify the analysis to determine the robustness.*

Assuming that the application tasks are assumed to be independent and irregular, the goal of the hierarchical DLS algorithms is scheduling these tasks onto the set of P processors (divided into disjoint groups) of the large-scale heterogeneous distributed system, while minimizing the total parallel execution time (or *makespan*) T_{PAR}. A minimum T_{PAR} is achieved via dynamic load balancing, using processor speeds (which in the case of AWF are periodically adapted) and hierarchical management. Each processor in a group executes a set of tasks (called chunk) at a time. Each task is executed in a non-preemptive fashion, i.e., no other tasks of higher priority will suspend it. The same holds for the execution of a chunk, or for all chunks during a single time-step. Table 1 summarizes the notations we used in the following sections. The performance features of interest are: $\mathcal{ET}_j$, T_{PAR}, and N_{resch}, and they should be limited in variation under certain application, system, or environment related parameters perturbations. For hierarchical DLS, perturbation parameters include variations in: irregularities of application computational requirements, *system availability due to unforeseen loads* (processors' delivered computational speed when shared among multiple users), network latency (delays in the communication speed due to network congestion), and *resource reliability* (caused by processor or network failures). Commonly, *all* perturbation parameters vary over time and cannot be accurately predicted before execution. A *robust hierarchical DLS algorithm* must adapt to any variations in these perturbation parameters, and yield performance parameters that vary in a constrained manner. Designing robustness metrics that incorporate *all* these parameters is very challenging [13].

2.1. Robustness against perturbations in system load

The parallel time given by a DLS method is generally defined by the processor with the longest individual finishing time (see Table 1). Assuming *unknown system load variations*, the individual finishing time $\mathcal{ET}_j$ of processor m_j must be robust against them. In other words, T_{PAR} be robust against such variations, as well. Assuming system load variations, the *actual* finishing time $\mathcal{ET}_j$ of processor m_j must be calculated considering the effects of errors in the estimation of the processor's load variation, and *must not* exceed $\tau_1(>1)$ times its estimated value $\mathcal{ET}_j^{orig}$, where τ_1 is a tolerance factor reflecting the robustness. The FePIA procedure for this analysis is outlined below.

S.1 Let $\Phi = \{\phi_1\}$, $\phi_1 = \mathcal{ET}_j$, $1 \leq j \leq P$ be the performance features set. The individual finishing time for all $\{\text{tasks } i | a_i \text{ executed on } m_j\}$, is:

N	total number of tasks
N^{resch}	# of tasks that need to be *rescheduled*
a_i	i-th task, $1 \leq i \leq N$
P	total number of processors
m_j	j-th processor, $1 \leq j \leq P$
T_j	*execution* time of task a_i on m_j
T_{ij}^{W2F}	*communication* time between m_j and its foreman for executing a_i
T_{ij}^{W2W}	*communication* time between m_j and any other workers for executing a_i (processor regrouping)
$\mathcal{ET}_j$	*finishing* time of all tasks computed by m_j
T_{PAR}	total parallel execution time, $T_{PAR} = \max(\mathcal{ET}_j, 1 \leq j \leq P)$
$\lambda = [\lambda_1 \ldots \lambda_P]^T$	vector of processors *load* (= system load)
$\mathbf{F} = [\mathbf{f}_1 \ldots \mathbf{F}_P]^T$	vector of processors status (active/*failed*) vector
$\Phi = \{\phi_1, \ldots\}$	set of *performance* features
$\Pi = \{\pi_1, \ldots\}$	set of *perturbation* parameters
τ_1, τ_2, τ_3	tolerance factors for performance features
$r_{DLS}(,)$	robustness radius
$\rho_{DLS}(,)$	robustness metric

Table 1. Notation

$\mathcal{ET}_j = \sum_{i,j}^{N,P} \left(T_j + T_j^{W2F} + T_j^{W2W}\right)$ (1)

S.2 Let the perturbation parameters set be $\Pi = \{\pi_1\}$, $\pi_1 = \lambda_j$, $1 \leq j \leq P$. We consider λ_j to be the *individual* load of processor m_j, and λ the vector that contains all processors load values. Initially, the DLS *assumes* that the system has λ^{orig} load, which can be usually determined by executing the first batch of chunks, as determined by the original factoring rules and their subsequent evolution. The j-th position in the λ^{orig} vector is the initial load of m_j.

S.3 The impact of λ_j over $\mathcal{ET}_j$, is determined by analyzing individually, for all processors, their finishing time given their *own* load. Each actual finishing time is expected to vary according to λ_j, denoted as $\mathcal{ET}_j(\lambda_j)$. Mathematically, for all {tasks $i|a_i$ executed on m_j under varying load λ_j}, this is written as:

$\mathcal{ET}_j(\lambda_j) = \sum_{i,j}^{N,P} \left(T_j(\lambda_j) + T_j^{W2F}(\lambda_j) + T_j^{W2W}(\lambda_j)\right)$ (2)

S.4 We must define the boundary values of $\pi_1 = \lambda_j$. First, we must decide whether the perturbation parameter is a continuous or a discrete variable. There are two ways to measure the load of a processor in heterogeneous systems: number of processes in the processor's run-queue [1] (discrete variable), or delivered processor speed, measured as percentage of processor availability [14] (continuous variable). In this work, λ_j is a *continuous* variable measuring processor availability, which is highly advantageous since it expresses the delivered processor speed, which in fact reflects simultaneously the impact of: applications' requirements, hardware capabilities, and network speed in one. The boundary values of λ_j must satisfy the following boundary relationships:

$\left\{\lambda_j \in \left\langle \lambda_j', \lambda_j'' \right\rangle \mid \left(f_1(\lambda_j') = \beta_1^{\min}\right) \wedge \left(f_1(\lambda_j'') = \beta_1^{\max}\right)\right\}$ (3)

The tolerable variation interval for $\mathcal{ET}_j$ (the performance feature of interest), is given by $\left\langle \beta_1^{\min}, \beta_1^{\max} \right\rangle$. The tolerable increase in the *actual* finishing time $\mathcal{ET}_j$ of processor m_j, considering the effects of errors in the estimation of variations of λ_j, cannot exceed $\tau_1(> 1)$ times its estimated value $\mathcal{ET}_j^{orig}$. The boundary relationships for this analysis are:

$\left\{\lambda_j \in \left\langle \lambda_j', \lambda_j'' \right\rangle \mid \left(\mathcal{ET}_j(\lambda_j) = \tau_1 \mathcal{ET}_j^{orig}\right) \wedge (1 \leq j \leq P)\right\}$ (4)

The robustness radius, $r_{DLS}(\mathcal{ET}_j, \lambda_j)$, is expressed as the *largest* increase in processor load, for any combination of processor load values, from the assumed value, that does *not* cause any tolerance interval violation for the execution time of all tasks a_i assigned to m_j. We must choose the norm which yield the smallest variation in the system (and ultimately processor) load, and we believe that a more intuitive norm to use is the ℓ_1-norm, and $r_{DLS}(\mathcal{ET}_j, \lambda_j)$ can be written as follows:

$r_{DLS}(\mathcal{ET}_j, \lambda_j) = \max \ \|\lambda_j - \lambda_j^{orig}\|_1 \ s.t. \ \mathcal{ET}_j(\lambda_j) = \tau_1 \mathcal{ET}_j^{orig}$ (5)

The *robustness metric* is the minimum of all robustness radii: $\rho_{DLS}(\Phi, \lambda_j) = \min\left(r_{DLS}(\phi_1, \lambda_j)\right) \ \forall \ \phi_i \in \Phi$ (6)

An acceptable value for τ_1 was proposed in [13] to be 1,2. For this analysis, $\rho_{DLS}(\mathcal{ET}_j, \lambda_j)$ is the general robustness metric of the "DLS" algorithm, with respect to each processor's individual finishing execution time against perturbations in the processor load, and $\rho_{DLS}(T_{PAR}, \lambda)$ is the robustness metric of T_{PAR} against variations in the total system load:

$\rho_{DLS}(T_{PAR}, \lambda) = \min(\rho_{DLS}(\mathcal{ET}_j, \lambda_j)), 1 \leq j \leq P$ (7).

2.2. Robustness against processor failures

Assuming an un-safe system with expected failures, N^{resch} and T_{PAR}, must both be robust against them: N^{resch} *must not* exceed τ_2% of the total number of tasks N, and T_{PAR} *must not* exceed $\tau_3(> 1)$ times it's estimated value T_{PAR}^{orig} (computed under the assumption that the system is completely safe).

When failures occur, the DLS algorithm must be able to reschedule the tasks that were assigned to the failed processors, as well as other tasks if necessary (for instance, to preserve load balancing on the remaining processors). To reduce the complexity of the analysis, we make the following simplifying assumptions: (i) only worker processors fail, (ii) failures occur simultaneously, and (iii) failures are permanent. We also assume that the DLS algorithm has fault-discovery and fault-recovery mechanisms. These assumptions can be relaxed in order to deliver more optimal and general robustness metrics. The FePIA procedure for this analysis is given below.

S.1 In this case the set of performance features has two elements: $\Phi = \{\phi_1, \phi_2\}$, $\phi_1 = N^{resch}$ and $\phi_2 = T_{PAR}$.

S.2 To identify the failing processors, we use consider $\mathbf{F} = [\mathbf{f}_1 \mathbf{f}_2 \ldots \mathbf{f}_P]^T$ as the vector containing the statuses of all processors, defined as $\mathbf{f}_j = 1$ if processor m_j failed, and $\mathbf{f}_j = 0$ otherwise, $1 \leq j \leq P$.

$\mathbf{F}^{orig} = [0 \ 0 \ldots 0]^T$, indicates that all processors are initially active. The perturbation parameters set is $\Pi = \{\pi_1 = \mathbf{F}\}$.

S.3 To determine the impact of Π over Φ, we need to determine separately each of the following:

$\phi_1 = f_{11}(\pi_1)$ $(8a)$ $\qquad \phi_2 = f_{21}(\pi_1)$ $(8b)$

which relate ϕ_1, and ϕ_2, respectively, to π_1. N^{resch} is directly proportional to the number of failing processors. Thus, $(8a)$ becomes $N^{resch}(\mathbf{F}) = N_p^{resch}(\mathbf{F}) + N_{lb}^{resch}(\mathbf{F})$ (9)
where $N_p^{resch}(\mathbf{F})$ is the total number of tasks assigned to the failed processors that need to be rescued (or restarted),

and $N_{lb}^{resch}(\mathbf{F})$ is the total number of 'surviving' tasks, assigned to 'surviving' processors, which the failure-recovery mechanism will need to reschedule together with $N_p^{resch}(\mathbf{F})$ with the goal of achieving and then maintaining a good load balancing on the remaining active processors. Additionally, $N_{lb}^{resch}(\mathbf{F})$ also depends on the choice of the DLS algorithm in use. It follows that the total parallel time T_{PAR} increases when processors start to fail. Hence, T_{PAR} is expected to vary with respect to $\mathbf{F}$ and relationship $(8b)$ can be written as $T_{PAR} = f_{21}(\mathbf{F})$. The exact impact of $\mathbf{F}$ over T_{PAR} depends on the choice of DLS algorithm and its fault-recovery mechanism.

S.4 To define the boundary values of $\pi_1 = \mathbf{F}$ for each element in Φ, we consider $\mathbf{F}$ a *discrete* variable that measures the number of "living" processors. We need to determine all the pairs of $\mathbf{F}$, such that for a given pair, the boundary value is the one that falls in the robustness region. Assume that $\mathbf{F}'$ is a perturbation parameter value, such that the machines that fail in the scenario represented by $\mathbf{F}'$ include the machines that fail in the scenario represented by $\mathbf{F}$ and *exactly one* other machine. Then, the boundary relationships are: $\left\{\mathbf{F}| \left(N^{resch}(\mathbf{F}) \leq \tau_2 N\right) \wedge \left(\exists \mathbf{F}' s.t.\ N^{resch}(\mathbf{F}') > \tau_2 N\right)\right\}$ (10)
$\left\{\mathbf{F}| \left(T_{PAR}(\mathbf{F}) \leq \tau_3 T_{PAR}^{orig}\right) \wedge \left(\exists \mathbf{F}' s.t.\ T_{PAR}(\mathbf{F}') > \tau_3 T_{PAR}^{orig}\right)\right\}$ (11)

where T_{PAR}^{orig} is the estimated parallel time assuming that the system is completely safe. We define the robustness radii for this case using the ℓ_1-norm: $r_{DLS}(N^{resch}, \mathbf{F}) = \max\ \|\mathbf{F} - \mathbf{F}^{orig}\|_1\ s.t.\ (N^{resch}(\mathbf{F}) \leq \tau_2 N)$
$\wedge(\exists \mathbf{F}' s.t.\ N^{resch}(\mathbf{F}') > \tau_2 N)$(12)
$r_{DLS}(T_{PAR}, \mathbf{F}) = \max\ \|\mathbf{F} - \mathbf{F}^{orig}\|_1\ s.t.\ (T_{PAR}(\mathbf{F}) \leq \tau_3 T_{PAR}^{orig})$
$\wedge(\exists \mathbf{F}' s.t.\ T_{PAR}(\mathbf{F}') > \tau_3 T_{PAR}^{orig})$(13)

$\rho_{DLS}(\Phi, \mathbf{F})$ is the robustness metric of the "DLS" algorithm against processor failures, with respect to N^{resch}, and T_{PAR}: $\rho_{DLS}(\Phi, \mathbf{F}) = \min\left(r_{DLS}(\phi_j, \mathbf{F})\right)\ \forall\ \phi_i \in \Phi$ (14)

3. Implementation and usefulness

An analysis of the computational complexity for computing such metrics based on the FePIA procedure is given in [13] (and references therein). The choice of τ_1, τ_2 and τ_3 impacts the robustness of DLS algorithms, and the proposed metrics are useful if these factors reflect reality with high accuracy. The metrics depend on certain application, system or algorithm specific parameters, most of which can be determined *apriori*. Hence, the metrics can be formulated offline and injected in the master to guide the dynamic scheduling process. If certain parameters become available (or known) only at runtime, the metrics are formulated using initial values (e.g., every element of vector $\mathbf{F}$ is zero, meaning no failed processors), which are updated in the master when newer values become available (e.g., certain processors failed, hence vector $\mathbf{F}$ contains non-zero elements). These metrics have no effect when no perturbations occur in the parameters against which they quantify the robustness of a DLS algorithm. However, they offer valuable information for making scheduling decisions when perturbations do occur in those particular parameters, leading to *feasible*, *qualitative* and *efficient* schedules.

4. Conclusions and future work

Scheduling today's applications on the latest computing platforms is challenging, and among other attributes, it must be realistic, efficient and robust. The metrics proposed in this work, in combination with the dynamic hierarchical management approach, are essential to bringing the most adaptive and efficient DLS algorithms to the state-of-the-art level required by today's computing platforms and applications. Immediate and future work directions include: devising similar robustness metrics for the adaptive DLS methods (adaptive factoring and recent variants of AWF), that use probabilistic analyses to model uncertainties; using multiple performance parameters and devise realistic robustness metrics that give proper weight to their impact over each performance features of interest; implementing these metrics and using them as performance metrics for evaluating the adaptive DLS methods in realistic large-scale platforms, individually against or in combination to traditional performance metrics, such as makespan, resource utilization, etc.

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

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

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

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

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

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

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

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

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

[11] Cariño, R. L., Banicescu, I.: 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] Riakiotakis, I., Ciorba, F. M., Andronikos, T., Papakonstantinou, G.: Self-Adapting Scheduling for Tasks with Dependencies in Stochastic Environments. Procs. Cluster Computing/HeteroPar '06 (2006)

[13] Ali, S., Siegel, H. J., Maciejewski, A. A.: Perspectives on Robust Resource Allocation for Heterogeneous Parallel and Distributed Systems. Chapter 4 of Handbook of Parallel Computing Models, Algorithms and Applications. MK Publishing (2008)

[14] Chtepen, M., Claeys, F.H.A., Dhoedt, B., De Turck, F., Demeester, P., Vanrolleghem, P.A.: Adaptive Task Checkpointing and Replication Toward Efficient Fault-Tolerant Grids. IEEE Trans. on Par. and Dist. Systems, 20:2, 180–190 (2009)