Adaptive Divisible Load Scheduling in Computational Grids

Luis de la Torre[1] Héctor de la Torre[2]
[1] Science and Technology School, Universidad Metropolitana, San Juan, PR
[2]School of Engineering, Turabo University, Gurabo, PR
Ana G. Mendez University System.
del1@suagm.edu, hectorfabio50@hotmail.com

Abstract

Several problems in science and engineering admit computational solutions that are implementable over Grid computing platforms. One problem, frequently faced by implementers is how to divide and distribute the workload into chunks among the Grid workers, the so-called load scheduling problem. Most commonly researches have studied this phenomena departing from a statics approach. This assumption is not fully functional in Grid environment where the resources are non-dedicated. This research proposed a methodology to integrate a statistic heterogeneous platform scheduling with a dynamic resources prediction to distribute the workload depending on future available resources. The SCOW algorithm is integrated to a tendency-based method, a mechanism to predict CPU utilization. These implementations can retro aliment the statistic scheduling algorithm to produce accuracy estimation to the Grid resources changes.

Kewords: Scheduling, grid computing, divisible load, divisible task, makespan, Throughput

1. Introduction

Divisible workload consists of workloads that can be partitioned into arbitrary tasks or chunks. These chunks in many cases are a core task that is repeated a number of times over different data. In single-program multiple-data (SPMD) style, these tasks are implemented as nested sequences of do-loops around the core task. Usually, a master process scheduler the chunks across all participating workers (round of data installments), so the execution time of the entire load (makespan) is minimum. In this research, is assumed that the distribution process use the network connection in a sequential fashion [3]. Each data installment is followed by a receive and a compute operation, both performed by the receiving worker. The p workers can compute and receive the next tasks concurrently. In SPMD implementations, rounds are controlled by an external do-loop, which imposes the periodic character of the job's execution. The main parameters in a SPMD implementation are thus, the number of rounds, denoted below by $\boldsymbol{m}$, the number of workers involved in the concurrent computations, denoted below by $\boldsymbol{q}$ and the chuck sizes $\boldsymbol{x}_i$ to the worker i, $1 \leq i \leq p$.
Two approaches dominate among the methodologies developed for scheduling of master-workers load. These methodologies are: steady state scheduling (SSS)[6, 7], and divisible load theory (DLT) [2, 3, 4.]. Both methodologies assume dedicated resources. These assumptions make the algorithm poor in real time environment such as Grid computing platforms with non-dedicated workers.

SCOW is a periodic user-level scheduler that tunes some selected parameters in a single-program multiple-data implementation of a master-worker parallel solution. SCOW minimizes the job make-span under either maximal production per period, or perfect worker utilization. This paper presents the theoretical foundations of SCOW to maximal production per period improving with the mixed tendency based strategy for predicting the CPU utilization of workers.

UMR [12] is a DLT multi-round algorithm for scheduling divisible loads on parallel computing systems. For homogeneous systems, the method uses uniform rounds meaning that in each round, each worker receives the same amount of work. There are two versions for the UMR idea. In the original, called UMR, the uniform amount of work is increased with each round. In a revised version, called UMR2, the uniform amount is increased or decreased depending on a parameter θ. If $\theta > 1$ the amount is increased, and decreased if $\theta < 1$. The UMR scheduler maintains perfect worker utilization throughout the execution, but if $\theta < 1$ there is no perfect worker utilization for URM2. Both methods model the system with a set of affine equations expressing execution times in terms of load. These equations are similar in spirit to the one used for SCOW. The model gives the amount of work for the first round and, through a recursive formula, the increments or decrements per round. Authors in [13] improve the UMR by predict the workers CPU utilization with a mixed tendency based strategy.

This paper is organized as follows: Section 2 discusses the model. Section 3 describes the Theoretical Foundations of SCOW. Section 4 shows the Makespan minimization problem to develop a multiround divisible load scheduling algorithm for affine cost models. In section 5, the local tasks CPU utilization in a CPU prediction strategy is incorporated. SCOWS was evaluated with extensive simulation in Section 6 and the executions is discuss later in on section 7.
Finally, Section 8 concludes the paper and discusses future directions.

2. Model

As stated in 1, is assumed a total number $\boldsymbol{X}$ of core tasks. These core tasks can be agglomerated to produce different chunks sizes (portion of job). These jobs are independent in the sense that neither ordering between them, nor synchronization among them is necessary.

2.1. Notation

As illustrated in Figure 1, the STARAFFINE network consists of p +1 processor, P= {P_0,P_1,P_2, ..., Pp}. The master processor is denoted P_0 while the p workers are labeled as P_i, $1 \leq i \leq p$. There are p communication links l_i from the master P_0 to each one of the workers Pi. Let x_i be the number of units of core tasks sent to worker P_i. l_i (x) measures the time units that takes for a load x to be moved from the master to the i^{th} worker in affine mapping model. Each worker i performs two operations, as well. These operations are message reception and the actual execution of the job, referred as computation. The worker i spends $w_i(x)$ time units in executing x core tasks, $w_i(x)$ is supposed to be an affine mapping.

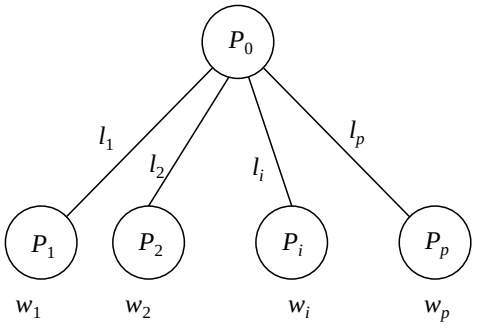

Figure1. Heterogeneous Star Graph

2.2. Architectural Model

As mentioned before, within a round, the master performs a sequence of data sends operations. Each data retrieval and send is followed by the data transmissions (l) over the network. Receive and compute (w) operations are performed by the workers upon the arrival of the data package. As a result, two major concurrent time segments are distinguished within a round: L, time spent by all network link in transmitting a round of data chunks; W maximum time spent by all workers in completing the reception of the data and execution of the corresponding data chunk. This research assumes the *full overlap, single-port model*. In this model, the master uses the network connection in a sequential fashion and the workers can perform the computation concurrently with data reception. One of the assumptions in this research is that the workers are non-dedicated processors. In Grid environmental the CPU power is distributed between local task and the Grid users.

2.3. Affine mapping

This subsection is a brief discussion of the affine maps in which the mathematical model is based. The execution times to each operations of data communication, and tasks execution vary as an affine mapping on the number of agglomerated core tasks x. This is,

$$l_i(x) = l_i \cdot x + L_i \quad (1)$$

$$w_i(x) = w_i \cdot x + W_i \quad (2)$$

for $1 \leq i \leq p$, where L_i is the initial cost of establishing a connection between the master P_0 and worker i, l_i is the send time associate to the data of a single core task; W_i is the overhead (startup time) of the computation in processor i and w_i corresponds to the execution time of a single core task.

3. Theoretical Foundations of SCOW

SCOW is designed as a periodic user-level scheduler for allocating agglomerated core tasks on parallel heterogeneous computing systems. This means that the mathematical framework behind SCOW is designed to return optimal constant values of the three parameter describe above $\boldsymbol{m}$, $\boldsymbol{q}$, and $\boldsymbol{x}_i$ to a master-worker SPMD implementation; under some specific constraints. Indeed, SCOW minimizes the make-span of the job under either maximal production per round or perfect system utilization [1]. In this research, the maximal production per period SCOW ability is selected, refers to a distribution of agglomerated core tasks across the workers that maximizes the total number of tasks completed in a round.

3.1. Maximal periodic production

In this section a brief description of scheduler theory is presented. The maximal production problem (MP) is a problem that imposes a restriction in the period to find the best approximation to the maximum number of task performed.

3.2. Problem

Suppose

$$m\sum_{i=1}^{p} x_i = X \tag{3}$$

where m is a number of round of data installments. The (MP) problem is stated as follows: Given a time period T, find a subset of $q+1$ workers such as

$$\textit{Maximize} \quad \sum_{i=1}^{q+1} x_i \tag{4}$$

$$\textit{Subject to} \quad L = \sum_{i=1}^{q+1} l_i(x_i) \le T \tag{5}$$

$$W = \max\{w_i(x_i)/1 \le i \le q+1\} \le T \tag{6}$$

$$x_i > 0,\ 1 \le i \le q+1 \tag{7}$$

3.3. Solution

Let MAXTASK(T) be the optimal solution of the previous problem. The next theorem provides a close form solution for the MP problem in homogeneous platform.

Theorem 1. *Let T be a real nonnegative numbers, $w_i=w$ and $l_i=l$ for all i and*

$$y = w^{-1}(T) \tag{9}$$

$$q = \lfloor T / l(y) \rfloor \tag{10}$$

$$T_\varepsilon = T - q \times l(y) \tag{11}$$

Then

$$\text{MAXTASK(T)} = qy + \max\{0, l^{-1}(T_\varepsilon)\} \tag{12}$$

The previous theorem has a possible extension to the heterogeneous problems. At this moment the best solution is an approximation theorem.

Theorem 2. *Let T be a real nonnegative number, p be a positive integer. The method:*

1. *Sort the workers by increasing communication times. Renumber them so that $l_1 \le l_2 \le \ldots \le l_p$.*
2. *Let $y_i=w^{-1}(T)$ for $1 \le i \le p$ and q the largest index so that $\sum_{i=1}^{q} l_i(y_i) \le T$. If $q<p$, let $T_\varepsilon = T - \sum_{i=1}^{q} l_i(y_i)$; otherwise let $T_\varepsilon=0$.*

Return the values to construct the following inequalities:

i. $$\left|\text{MAXTASK}(T) - \sum_{i=1}^{q} y_i + \max\{0, l_{q+1}^{-1}(T_\varepsilon)\}\right| \le \frac{\sum_{i=1}^{q+1} L_i}{l_{q+1}}$$

ii.If the period T is large enough

$$\left|\text{MAXTASK}(T) - \sum_{i=1}^{q} y_i + \max\{0, l_{q+1}^{-1}(T_\varepsilon)\}\right| \le \frac{L_{q+1}}{l_{q+1}}$$

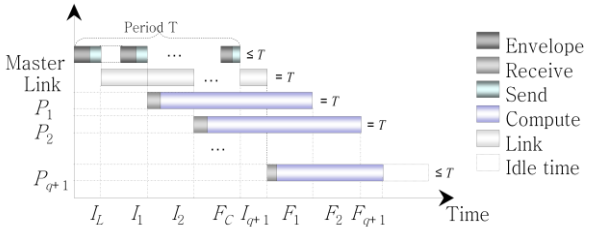

Figure 2. Gantt Char interpretation to the proposes solution

The theorem 2 gives an approximate to the optimal solution to the MP problem. In section 4 this approximation is used as a restriction to formulate a makespan minimization problem.

Figure 2 is a graphical representation to the solution given by the theorem 2. This solution follows the principle of bandwidth-centric [3, 4] because the priorities do not depend on the workers computation capabilities, only on their communication capabilities. Elsewhere, the number of select processor is directly affected by the computation capacity. In Grid computing the computational capacity depends on the local CPU utilization. This tendency is estimated using prediction strategy described in section 6

3.4. Last round modification

A common condition to get an optimal schedule is that all processor finishes the work at the same time. Modifications of the last round are used to impose the condition that all processors end operating at the same time[1]. This last round modification introduces a constant makespan reduction of $1/2T$.

4. Makespan minimization

The makespan minimization problem constrained to maximal production (MMP-MP) solution approximation and the workers order described in theorem 2 is formulate as follow:

Minimize $\mu(T) = (M+1/2)T + l_1(Y)$

$$\textit{subject to} \quad \text{X} = v\left(\sum_{i=1}^{q} y_i + y_{q+1}\right) \tag{13}$$

$$T = w_i(y_i), \qquad 1 \le i \le q \tag{14}$$

$$T = \sum_{i=1}^{q+1} l_i(y_i) \tag{15}$$

$$T \ge w_{q+1}(y_{q+1}) \tag{16}$$

$$y_i \ge 0, \qquad 1 \le i \le q \tag{17}$$

$$p \ge q+1 \tag{18}$$

The problem is solved by Lagrange multipliers [9]. This solution is stated in the next theorem.

Theorem 3: *Let X be a nonnegative real number and q a positive integer (q+1 ≤ p) . Then the solution to the MMP-MP problem without restriction* 16 *with q+1 processor is,*

$$T = \frac{1}{a(q)}\sqrt{\frac{b(q)w_1X}{\frac{1}{2}w_1 + l_1}} + \frac{b(q)}{a(q)} \qquad (19)$$

where $$a(q) = \sum_{j=1}^{q+1}\frac{1}{w_j} + \frac{1}{l_{q+1}}\left(1 - \sum_{j=1}^{q+1}\frac{l_j}{w_j}\right) \qquad (20)$$

and $$b(q) = \sum_{j=1}^{q+1}\frac{W_j}{w_j} + \frac{1}{l_{q+1}}\left(\sum_{j=1}^{q+1}L_j - \sum_{j=1}^{q+1}\frac{l_jW_j}{w_j}\right) \qquad (19)$$

the theorem 3 permit reformulates the MMP-MP, in the problem to finding the minimal value of $\mu(T)$ with *i* ranging over the subset the *i* that satisfying

$$w_{i+1}(y_{i+1}) \le T \qquad (20)$$

5. CPU prediction strategy

In Grid computing typically the resources are non-dedicated, that is, the availability of the full processing speed is no guaranteed. Let *S*, the full processor speed. The local task execution generates a CPU utilization, if the *Utilization* can be predicted them the *ActualSpeed* can be computed as follows:

ActualSpeed = *S* * (100%-*Utilization*) (21)

This *ActualSpeed* is used as retro-alimentation information to the static scheduler. To predict the CPU load and utilization is used a time series prediction approach [10, 11] that has been probed the effective empirically.

The idea of this prediction strategy is based on the assumption that if the current value increases, the next value will also increase, and if the current value decreases, the next value will also decrease.
Formally, we can write:

```
If (U_{T-1} < U_T)
     IncrementValueAdaptation()
   P_{T+1} = U_T + IncrementValue
Else if(U_{T-1} > U_T)
   DecrementFactorAdaptation()
   P_{T+1} = U_T × IncrementValue
```

Where,
U_T: the measured utilization at measurement T,
P_{T+1}: the predicted utilization for measurement U_{T+1},
H: the number of historical data points used in the prediction.

Increment value and decrement factor can be calculated as:

```
Procedure: INNCREMENTVALUEADAPTATION()
Mean = (1/n)∑i
RealIncValue =U_T − U_{T-1} ;
NormalInc = IncrementValue + (RealIncValue −
           IncrementValue) × AdaptDegree;
if (U_T < Mean)
  IncrementValue = NormalInc;
Else
   PastGreater = (number of data points>UT)/H
   TurningPointInc=IncrementValue×PastGreater
   IncrementValue=Min(NormalInc,
                  TurningPointInc)
```

AdaptDegree can range from 0 to 1 and expresses the adaptation degree of the variation. The best values for input parameters such as AdaptDegree and DecrementFactor are determined empirically.

6. Experiment

In table 1 the group of numerical values selected to perform the simulation is presented. The values are used to predict the corresponding SCOW-MP and UMR version develop in [13]. These predictions are made using the mathematical equations underlying them and a random variable to CPU simulation is also generated using gamma function

Table 1: Simulation Parameter

Parameter	Values
Number of workers	p=10,20,30,40
Agglomerated tasks	X = 1000
Computational rate	S_i=.5+*randonvariavle* $w_i = 1/S_i$
randonvariavle	is also generated using gamma function with fixed arrival time landa=.5 and beta = 1
Transfer rate	B_i = 1.1p to 1.1p + 1; l_i =1/b
Computational latency	$cLat_i$ = 0:03; W_i = cLat;
Communication latency	$nLat_i$ = 0:03; L_i = nLat

7. Result

It is worth remarking that UMR methods and the optimal number of rounds, and perform no discretization on the amount of work per round. Thus, in order to make the comparisons possible, SCOW discretizations are made on the number of rounds and not on the amount of agglomerated tasks per round. The randomly chosen values are shown in Table1.

The numerical prediction of the performances of UMR and SCOW are shown in Table 2.

Table 2: Comparison Between SCOW and UMR

	SCOW-MP	UMR	UMR2
Normalized Make-span	1.000	1.012	1.032
Normalized Workers CPU Utilization	1.000	1.008	1.009

Table 2 shows the comparison between SCOW-MP, UMR and UMR2, averaged over similar (in the number of workers) experiment. All the scheduler was improved by a last round modification in order keep consistent the comparison.
The first row shows the ratio of make-span achieved for the 3 schedulers. The second row shows the similar ratio for the system utilization, but at this time the ratio is inverted, because the maximal values is the best. The main observation is that SCOW-MP outperforms UMR and UMR2 on average. The SCOW-MP is the best algorithm in Make-span and system utilization.

8. Conclusions

Many researches in maximal throughput in lineal model can be found in the literature. These results are developed to the problem formulated for fixed sizes tasks. In this research the problem is exported to affine model and in contrast the goal is maximize the production, that is, the total number of tasks processed.

The contribution of this research includes an optimal solution in the homogeneous case and approximate solution in the heterogeneous case, integrate with a CPU prediction strategy to perform a scheduler reliable in a Grid environment. The makespan and system utilization of two algorithms is also compared.

The results show that the proposed SCOW-MP algorithms outperform the competitors. Future work includes the development of a strategy to predict the network utilization; due to SCOW is a bandwidth-centric algorithm.

9. Acknowledgments

This work was made possible by funding from the Caribbean Computer Center of Excellence (CCCE) under NSF Award number CNS-0940522.
Thanks are due to Dr. Juan F Arratia and Dr. Oliva Primera-Pedrozo from the Universidad Metropolitana-Cupey for their support.

References

1. L. de la Torre, “Scheduling divisible tasks under production or utilization constraints”, PhD diss., University of Puerto Rico, Mayaguez, Puerto Rico 2010.
2. Y. Yang, K. van der Raadt, H. Casanove, Multiround Algorithms for Scheduling Divisible Loads, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 16, No. 11, 2005.
3. C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand and Y. Robert, Scheduling Strategies for Master-slave Tasking on Heterogeneous Processor Platforms, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 4, pp. 319-330, 2004.
4. M. Drozdowski and P. Wolniewicz Optimum Divisible Load Scheduling on Heterogeneous Stars with Limited Memory, *European Journal of Operation Research*, Vol. 172, No. 2, 2006.
5. N. Jones and P. Pevzner An Introduction to Bioinformatics Algorithms, MIT Press, 2000.
6. V. Bharadwaj, D. Ghose, V. Mani, and T.G. Robertazzi. Scheduling Divisible Loads in Parallel and Distributed Systems. *IEEE Computer Society Press*, 1996.
7. O. Beaumont, H. Casanova, A. Legrand, Y. Robert, Y. Yang: Scheduling Divisible Loads on Star and Tree Networks: Results and Open Problems. *IEEE Trans. on Parallel and Distributed Systems*, vol. 16, no. 3, 2005, 207-218.
8. D. Bertsekas. Constrained Optimization and Lagrange Multiplier Methods. *Athena Scientific, Belmont, Mass.*, 1996.
9. D. Bertsekas, editor. Constrained Optimization and Lagrange Multiplier Methods. Athena Scienti_c, Belmont, Mass., 1996.
10. L. Yang, J.M. Schopf, and I. Foster, Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decision in Dynamic Environments, *SuperComputing 2003*, Phoenix, Arizona USA November 2003.
11. L.Yang, I. Foster, and J.M. Schopf, Homeostatic and Tendency-Based CPU Load Predictions, *International Parallel and Distributed Processing Symposium (IPDPS'03)*, Nice,France, April 2003.
12. Yang Yang and Henri Casanova, UMR: A Multi-Round Algorithm for Scheduling Divisible Workloads; *Proceeding of the International Parallel and Distributed Processing Symposium* (IPDPS’03), Nice, France, April 2003.
13. Said Elnaffar and Nguyen The Loc. "Enabling Dynamic Scheduling in Computational Grids by Predicting CPU Utilization"; WSEAS Transactions on Communications Issue 12, Volume 4, pages 1419-1426. December 2005.