A Methodology for Dynamic Scheduling of Divisible Workloads in Grid Environments

SAID ELNAFFAR‡ and NGUYEN THE LOC†
‡College of Information Technology, UAE University, Al-Ain, UAE
elnaffar@uaeu.ac.ae http://faculty.uaeu.ac.ae/elnaffar
†Japan Advance Institute of Science and Technology, Japan

Abstract: Scheduling divisible workloads in distributed systems has been one of the interesting research problems over the last few years. Most of the scheduling algorithms previously introduced are based on the master-worker paradigm. However, the majority of these algorithms assume that workers are dedicated machines, which is a wrong assumption in distributed environments such as Grids. In this work, we propose a dynamic scheduling methodology that takes into account the two prominent aspects of Grids: heterogeneity and dynamicity. The premise of our methodology is to use a prediction strategy to estimate the CPU speed of each Grid resource and subsequently feed this estimation to a static scheduling algorithm in order to divide the workload into suitable chunks in light of the available computational power. Such integration can produce dynamic scheduling algorithms that can handle the constantly changing properties of Grid resources.

Key-Words: Grid Computing, Divisible Workload, Dynamic Scheduling Algorithm, CPU Speed Prediction.

1 Introduction

Grid computing is the computation model that combines all distributed computing resources and allocates them as needed for applications [20]. A critical issue for the performance of a Grid is the task-scheduling problem, that is, the problem of how to divide an application workload into many parts and assign them to computers of the Grid, here thereafter called *workers*, so that the execution time or *makespan* is minimum.

There are many algorithms [1, 2, 3, 7] for scheduling divisible workloads (workloads that can be partitioned by the scheduler into arbitrary tasks or 'chunks') that assume that computational resources are dedicated. This assumption renders these algorithms impractical in distributed environments such as Grids where computational resources are expected to serve local tasks in addition to the Grid tasks (i.e., non-dedicated workers). Another shortcoming of these algorithms is that they do not take the dynamicity of Grids into account. In reality, the CPU and bandwidth utilization of workers vary over time. An efficient scheduling algorithm should factor in such changes in CPU and bandwidth capacity. Our main contribution in this work is a dynamic scheduling methodology that takes into account the heterogeneity of computational powers and their dynamicity over time.

As depicted in Fig. 1, the Grid application (master) submits its divisible workload [7] to a scheduler for processing. In order for the scheduler to decide how to divide up the workload and decide on the size of the chuncks to be disseminated to each worker, it needs to know the available computational power (CPU speed) of each worker. Therefore, the scheduler queries a prediction component that performs short-term forecasting by collecting and analyzing a series of CPU utilization values. The modularity of this approach provides the flexibility of trying any scheduling algorithm integrated with any prediction mechanism.

The rest of the paper is organized as follows. Section 2 reviews some of the static and dynamic scheduling algorithms. Section 3 describes our heterogeneous computation platform. Section 4 briefly describes the static scheduling algorithm used in this work. The prediction mechanism that we use to predict the processor speed is explained in Section 5. We conclude and sketch future work in Section 6.

2 Related work

Single round algorithms [3, 15] are the early and simple way of scheduling and sharing workloads among workers. As shown in [3], for a large workload, the single-round approach is inefficient due to the idleness that the last worker incurs until it receives its chunk. The first multi-round scheduling algorithm was introduced in [7] but computation and communication startup costs were overlooked, which made this algorithm less realistic.

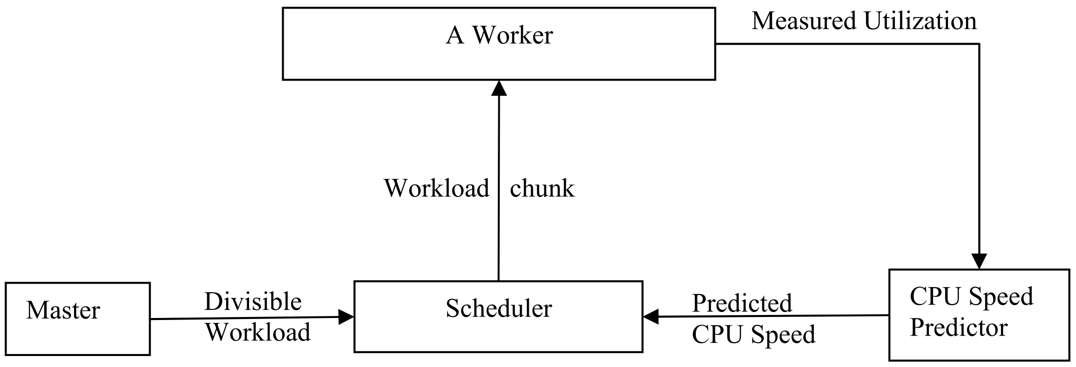

Fig. 1. Predicting worker's performance helps scheduling algorithms be dynamic

Some studies in distributed systems [1, 2, 3, 8] focus on affine models that account for computation and communication startup times. However, these algorithms are deemed static scheduling algorithms as they assume fixed and guaranteed availability of computational power of Grid workers, which is an impractical assumption because workers are typically non-dedicated processors. The RUMR algorithm [8] is designed to tolerate performance prediction errors by using the Factoring technique [15], however, all of the parameters are initially set and remain fixed throughout the scheduling process, which makes RUMR non-adaptive.

Some researches such as [5, 6] have been aware of the importance of capturing the dynamic aspects of resources. However, they do not integrate their dynamic models with scheduling algorithms. In [4], the author uses an M/M/1 system to model the performance of a non-dedicated Grid worker as it processes local and Grid applications. However, this work does not handle divisible workloads.

Our work presents a methodology in which we can predict the computational capacity of Grid workers and feed this information to a scheduling algorithm in order to process divisible workloads.

3 Workload and Platform Models

Our methodology assumes the master-worker paradigm for the Computational Grid. The platform topology we deal with consists of heterogeneous workers (processors) connected to a master by heterogeneous network links. We consider a Grid application (master) that generates a divisible workload [7], W_{total}, that needs to be split into chunks and disseminated to N workers for processing. One of the important assumptions in our computational model is that workers are non-dedicated processors; an assumption which is closer to reality with respect to Grid environments. A worker should process local tasks as well as external Grid tasks. Consequently, the computational power of a worker available to Grid tasks may vary over time due to the competition with local tasks. We assume that the time needed to process a chunk is proportional to its size. We assume that the master does not send chunks to workers simultaneously, although some pipelining of communication can occur [17]. An initial investigation of simultaneous transfers (e.g., WAN) is presented in [1]. We also assume that a worker can receive data from the network and perform computation simultaneously.

We model the time required to process the workload $chunk_i$ on $worker_i$, $1 \le i \le N$, as

$$Tcomp_i = cLat_i + \frac{chunk_i}{ES_i}$$

where $cLat_i$ is an initial overhead, in seconds, incurred by the processor, and ES_i is the estimated computational speed of the worker in units of workload performed per second. The communication time of sending $chunk_i$ to $worker_i$ is

$$Tcomm_i = nLat_i + \frac{chunk_i}{B_i} + tLat_i$$

where $nLat_i$ is the initial cost (in seconds) of establishing a connection between the master and worker i; B_i is $worker_i$'s bandwidth measured in units of workload per second; $tLat_i$ is the post cost (in seconds) of terminating the connection (the master finishes pushing data on the network to worker i plus the time when worker i receives the last byte of data).

We assume that the $nLat_i$ and $chunk_i/B_i$ portions of the transfer are not overlappable with other data transfer. However, $tLat_i$ is overlappable. This model is flexible and can accommodate different computational scenarios used in previous research related to scheduling divisible workloads.

4 The UMR Scheduling Algorithm

Our work is an augmentation to the static UMR (Uniform Multi-Round) algorithm that is explained in detail in [1, 2]. The UMR algorithm outperforms its two competitors, the Multi-installment (MI) [7] and One-Batch [17] algorithms, in an overwhelming majority of the cases. Here we briefly describe the UMR algorithm and explain how we propose to make it dynamic. Fig. 2 shows how UMR dispatches chunks of workloads in multiple rounds, so we have

$$W_{total} = \sum_{i=1}^{M} round_j$$

$$round_j = \sum_{i=1}^{N} chunk_{j,i}$$

where

- $round_j$: amount of workload the master delivers during round j,

- $chunk_{j,i}$: the fraction of the total workload, W_{total} , that the master delivers to worker i in round j ($1 \le i \le N$; $1 \le j \le M$). M is the number of rounds required to dispatch all chunks to workers.

The generic heterogeneous version of UMR splits the workload into chunks in such a way that each worker in $round_j$ finishes its computation in a constant time, $const_j$. That is:

$$cLat_i + \frac{chunk_{j,i}}{ES_i} = const_j (\forall i = 1..N)$$

By combining the last two equations, we obtain a simple induction relation on the chunk sizes,

$$chunk_{j,i} = \alpha_i \times round_j + \beta_i$$

such that

$$\alpha_i = \frac{ES_i}{\sum_{k=1}^{N} ES_k}$$

$$\beta_i = \frac{ES_i}{\sum_{k=1}^{N} ES_k} \sum_{k=1}^{N} \left(ES_k \times cLat_k \right) - ES_i \times cLat_i$$

where M (number of rounds) and $chunk_0$ are unknown. ES_i is the estimated speed of worker i. Typically, this speed fluctuates and varies over time. In order to handle this dynamicity, we predict the available CPU speed as explained in the next section. In light of the predicted CPU speed for each worker, the scheduling algorithm determines the sizes of workload chunks in each round.

The next step in the UMR development is that we frame the problem as a constrained optimization problem: the objective is to minimize the makespan (total execution time) of a Grid application subject to the constraint that all the chunks sum up to the total workload. Using the Lagrange Multiplier method [18] we can obtain a system of two equations with M and $chunk_0$ as unknowns. Full details of the solution for this optimization problem can be found in [2].

5 Worker Speed Prediction

Most static scheduling algorithms [1, 2, 3, 7] rely on accurate estimation of the execution time of a task at a worker based on the assumption that the worker is a dedicated machine. This assumption is usually unrealistic in a Grid environment where workers are responsible for executing their local tasks and, if they become underutilized, they can handle incoming Grid tasks. Typically, the priority is given to local tasks. Consequently, and depending on the local load, we cannot always assume the availability of the full processing speed, S, to Grid tasks. Based on the measured CPU utilization, the *ActualSpeed* that is available for Grid tasks can be computed as follows:

$$ActualSpeed = S * (100\%\text{-}Utilization)$$

Therefore, if we predict the *Utilization* of a worker, we can compute and send the anticipated processing speed (*ES*) to the scheduling algorithm. In this section, we use a time series prediction approach that has been empirically effective in predicting CPU load and utilization [5, 6]. It predicts a one-step-ahead value of utilization based on a fixed number of immediately preceding historical data measured at a constant-width time interval. We use the following notation:

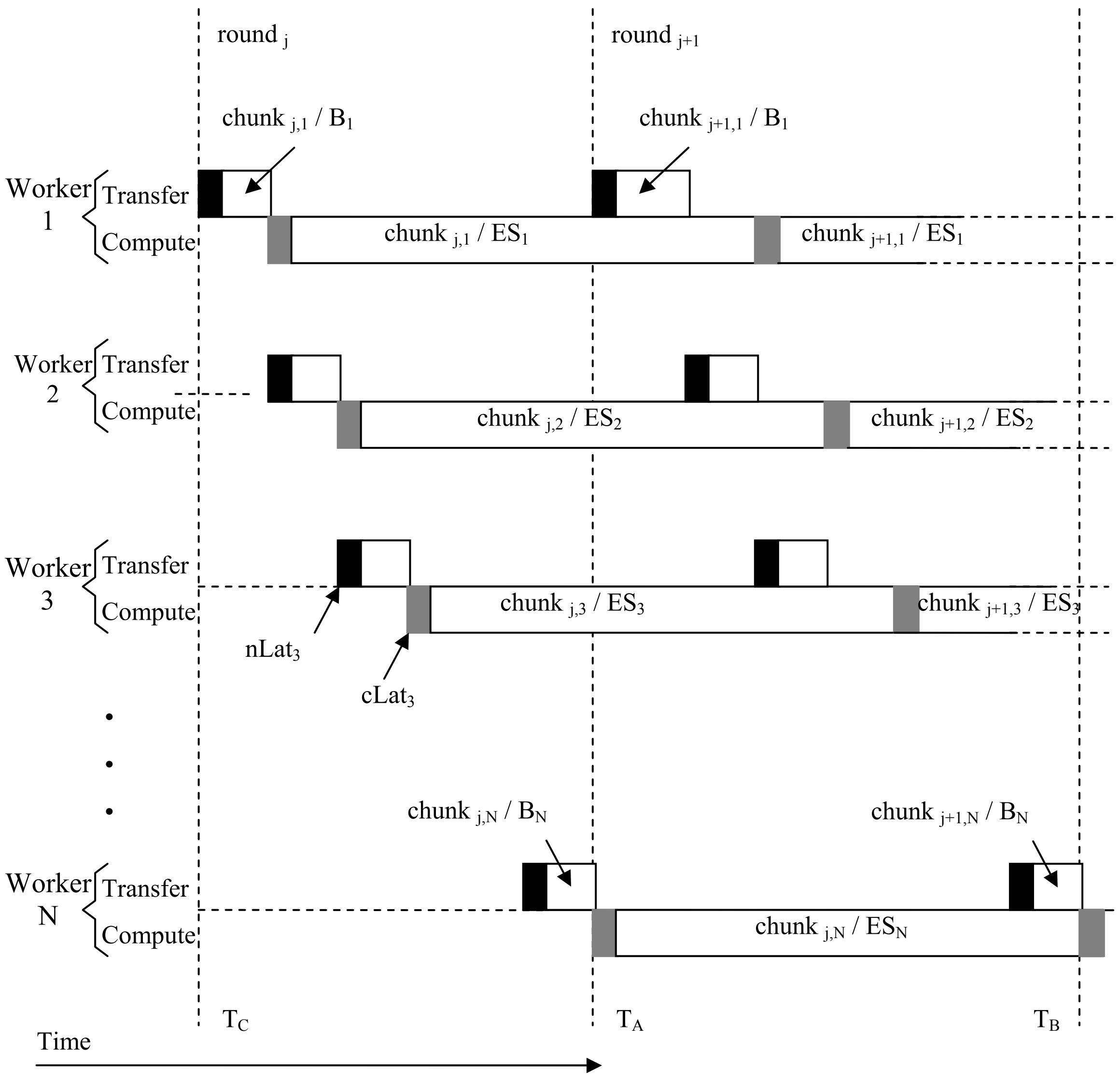

Fig. 2. UMR dispatches workload chunks in rounds

U_T: the measured utilization at measurement T,

P_{T+1}: the predicted utilization for measurement value U_{T+1},

H: the number of historical data points used in the prediction, also called the window size.

Mixed tendency-based prediction strategy

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 ($U_{T-1} < U_T$)            // Tendency is increase
{
        IncrementValue Adaptation process;
        $P_{T+1} = U_T + IncrementValue$;
}
else if ($U_{T-1} > U_T$)
{       // Tendency is decrease
        DecrementFactor Adaptation process;
        $P_{T+1} = U_T \times DecrementFactor$;
}
```

The tendency-based strategy has a possible source of error when the time series is going to a turning point and changes its direction, that is, when an increasing time series becomes decreasing one, or vice versa. To minimize this kind of error, we use the *Mean* as the *threshold value*. Let us consider the increasing time series, if the current value U_{T+1} is smaller than the threshold value, the variation will be adapted normally. If U_{T+1} is bigger than the threshold value, it is possible that the next step is a *turning point*. We calculate the value of *PastGreater* by the percentage of the history data that is greater than U_T and use this value as the possibility of the event that T is not a turning point.

The adaptation process in both of Increase and Decrease are similar. For example, the adaptation process for the IncrementValue can be described as follows

```
$Mean = (1/n) \times \sum_{i=1}^{n} i$
$RealIncValue = U_T - U_{T-1}$ ;
$NormalInc = IncrementValue + (RealIncValue - IncrementValue) \times AdaptDegree$;
if ($U_T < Mean$)              //Normal adaptation
        $IncrementValue = NormalInc$;
Else {
   $PastGreater$ = (number of past data points > $U_T$) / $H$;
   $TurningPointInc = IncrementValue \times PastGreater$;
   $IncrementValue = Min(NormalInc, TurningPointInc)$;
}
```

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 Conclusion and Future Work

In this paper we propose a methodology that can render a static scheduling algorithm dynamic by augmenting it with a prediction component that can forecast how the characteristics, such as the computational power, of a Grid resource change over time. Based on the estimated performance of Grid resources, the scheduling algorithm can decide how to divide and distribute workload chunks. We use the UMR as the static scheduling algorithm and the tendency-based time series prediction as a prediction method. In the majority of experiments the performance of UMR is superior to its competitors [1]. The tendency-based prediction method manifests its success empirically too [5, 6]. We extrapolate that their integration can lead to a similar success with respect to capturing the dynamicity of the Grid. We are presently in the experimentation phase. In addition to the artificially generated data, we will use trace-driven simulation to validate our methodology. Real data will be obtained from the Network Weather Service (NWS) [19].

As a sketch of future work, it is interesting to:

- develop models that allow the master to establish simultaneous connections with the workers (e.g. WAN).

- incorporate the cost of shipping the results from the workers back to the master. Most of the existing studies [1, 2, 3, 7, 8], including ours, assume this cost negligible.

- study the performance of the Grid application when the computational cost is not directly proportional to the size of its workload.

- predict changes in the bandwidth of each Grid worker and pass on such information to the scheduler.

Finally, the proposed methodology is flexible enough to incorporate different static scheduling algorithms (e.g., [1, 2, 3]) with different prediction techniques (of various degrees of complexity). Therefore, we would like to experiment with a number of scheduling algorithms that collaborate with different types of predictors.

Acknowledgement

We would like to express our sincere appreciation to Yang Yang for his cooperation with respect to the UMR algorithm. We also would like to thank Wendy Powley for proofreading our paper.

References:

[1]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.

[2] Yang Yang and Henri Casanova, A Multi-Round Algorithm for Scheduling Divisible Workload Applications: Analysis and Experimental Evaluation, *Technical Report CS2002-0721*, Dept. of Computer Science and Engineering, University of California, 2002.

[3] O. Beaumont, A. Legrand, and Y. Robert, Scheduling Divisible Workloads on

Heterogeneous Platforms, *Parallel Computing*, Volume 29 , Issue 9 September 2003.

[4]Y. Zhang, Y. Inoguchi, and H. Shen, A Dynamic Task Scheduling Algorithm for Grid Computing System, *Second International Symposium on Parallel and Distributed Processing and Applications (ISPA'2004)*, Hong Kong Dec. 2004.

[5] 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.

[6] 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.

[7] V. Bharadwaj, D. Ghose, V. Mani, and T.G. Robertazzi, Scheduling Divisible Loads in Parallel and Distributed Systems, *IEEE Computer Society Press*, 1996.

[8] Yang Yang and Henri Casanova, RUMR: Robust Scheduling for Divisible Workloads, *12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, Seattle, Washington, USA 2003.

[9] A. Papoulis and S.U. Pillai, *Probbility, Random Variables and Stochastic Processes*, Fourth Edition McGraw-Hill , 2002.

[10] R. Wolski, Dynamically Forecasting Network Performance Using the Network Weather Service, *Journal of Cluster Computing*, 1998

[11] H.J. Dail, A Modular Framework for Adaptive Scheduling in Grid Application Development Environments, *Computer Science*, University of California, California, San Diego, 2001.

[12] C. Liu, L. Yang, I. Foster and D. Angulo, Design and Evaluation of a Resource Selection Framework for Grid Applications, *11th IEEE International Symposium on High-Performance Distributed Computing (HPDC 11)*, Edinburgh, Scotland, 2002.

[13] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, Heuristics for Scheduling Parameter Sweep Applications in Grid Environments, *Proceeding of the 9th Heterogeneous Computing Workshop (HCW'2000)*, May 2000.

[14] T. Braun, H. Siegel, and N.Beck, A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems, *Journal of Parallel and Distributed Computing*, 61:810-837, 2001.

[15] J. Błazewicz, M. Drozdowski, and M. Markiewicz, Divisible Task Scheduling-Concept and Verification, *Parallel Computing*, 25:87-98, 1999.

[16] S. Flynn Hummel, Factoring : a Method for Scheduling Parallel Loops, *Communications of the ACM*, 35(8):90–101, August 1992.

[17] A. L. Rosenberg, Sharing Partitionable Workloads in Heterogeneous NOWs: Greedier Is Not Better, *Proceedings of the 3rd IEEE International Conference on Cluster Computing (Cluster 2001)*, pages 124–131,2001.

[18] D. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*, Athena Scientific, Belmont, Mass., 1996.

[19] R. Wolski, N. Spring, and J. Hayes, The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, *Journal of Future Generation Computing Systems*, pp. 757-768, 1998.

[20] I. Foster and C. Kesselman (eds.), *The Grid 2: Blueprint for a new Computing Infrastructure*, Morgan Kaufmann, 2004.