GENETIC-BASED SOLUTIONS FOR INDEPENDENT BATCH SCHEDULING IN DATA GRIDS

Joanna Kołodziej
Cracow University of Technology, Poland
Email: jokolodziej@pk.edu.pl

Magdalena Szmajduch
CDN Partner Cracow, Poland
E-mail: magdalena.szmajduch@cdnpartner.pl

Samee U. Khan
North Dakota State University
Fargo, USA
Email: samee.khan@ndsu.edu

Lizhe Wang
Center for Earth Observation and Digital Earth
Chinese Academy of Sciences
Beijing, China
Email: LZWang@ceode.ac.cn

Dan Chen
China University of Geosciences
Wuhan, China
E-mail:Danjj43@gmail.com

KEYWORDS

Data Grid, Scheduling, Data Center, Expected Time to Transmit, Data replication, Genetic Algorithm

ABSTRACT

Scheduling in traditional distributed systems has been mainly studied for system performance parameters without data transmission requirements. With the emergence of Data Grids (DGs) and Data Centers, data-aware scheduling has become a major research issue. In this work we present two implementations of classical genetic-based data-aware schedulers of independent tasks submitted to the grid environment. The results of a simple empirical analysis confirm the high effectiveness of the genetic algorithms in solving very complex data intensive combinatorial optimization problems.

INTRODUCTION

In today's modern heterogeneous computational systems with massive data processing, data-aware scheduling is one of the crucial problem, which has attracted considerable attention of researchers in data intensive computing. Much of the current efforts are focused on scheduling tasks workloads, data location reorganization [8] and energy-effective scheduling in large-scale data centers [4]. In many grid and cloud approaches, the scheduling problems are divided into two main classes: (i) those, which can be solved in computational systems, where usually it is assumed that data is delivered a priori and no data transfer times, data access rights, data availability (replication) and security issues are considered; and (ii) those, which can be solved just in Data Grids or data centers. However, efficient grid or cloud schedulers must take into account the features of both computing and data infrastructures to achieve desired performance of grid-enabled applications [7]. In such systems the data hosts are usually distributed in similar way as the computational nodes, which makes the general scheduling problem a real research challenge [3].

In this work, we address a general grid scheduling problem of data intensive applications submitted independently by the grid end users. Based on our previous work [6], we have integrated the data transmission and data nodes location criteria with the traditional scheduling objectives, namely makespan and flowtime. We provided a simple empirical analysis with genetic-based schedulers, that have been also tested in our previous works for similar class of problems, where data access and processing were ignored (see [5] for details). This analysis confirms a high effectiveness of genetic-based

Proceedings 27th European Conference on Modelling and Simulation ©ECMS Webjørn Rekdalsbakken, Robin T. Bye, Houxiang Zhang (Editors)
ISBN: 978-0-9564944-6-7 / ISBN: 978-0-9564944-7-4 (CD)

schedulers in solving complex data-intensive combinatorial optimization problems in the dynamic computational environments. All the experiments have been conducted by using *Data-Sim-G Batch* data-aware grid simulator developed by the authors.

The remainder of this paper is structured as follows. First we define a modified Expected Time to Compute matrix model for data-aware independent batch scheduling. A brief presentation of the genetic schedulers and main concept of *Data-Sim-G Batch* grid simulator is followed by a simple analysis of the experiments conducted for two variants of the genetic schedulers. The paper ends with simple conclusions and future research plan.

DATA-AWARE SCHEDULING IN THE GRID SYSTEM

Data-aware ETC Matrix model

We consider in this paper a general batch scheduling problem of tasks independently submitted to the system by the data-grid end users. This problem can be defined by the following four components (see also [6]):

- a batch of grid applications (tasks) $N_{batch} = \{t_1, \ldots, t_n\}$, where n - is the size of the batch (the number of tasks in the batch);
- a set of computational grid resources $M_{batch} = \{m_1, \ldots, m_k\}$, ($k$ - is the total number of machines available in the system for a given batch;
- a set of data-files $F_{batch} = \{f_1, \ldots, f_r\}$ needed for the completion of the tasks from N_{batch}; and
- a set of data-hosts $DH = \{dh_1, \ldots, dh_s\}$ with the necessary data service capabilities.

We assume that 'tasks' in our model can be complex data-intensive applications, and 'machines' can be single CPUs, parallel machines or even small local computing clusters. Those applications require multiple data files from data hosts, which can be also distributed in the grid system. It means that data files needed for completing the grid applications can be located (and/or replicated) at various grid nodes and their transfer to the computational nodes can be provided by the networks of varying capability.

For the characteristics of tasks in the batch, we introduce a *batch workload vector* $WLoad_{batch} = [wload_1, \ldots, wload_n]$, where $wload_j$ denotes an estimation of the computational load of a task t_j (in Millions of Instructions –MI). Each task t_j requires a set of files $F_j = \{f_{(1,j)}, \ldots, f_{(r,j)}\}$ ($F_j \subseteq F_{batch}$) that are distributed on a subset DH_j of the data nodes DH. We assume that each data host can serve multiple data files at a time and data replication is *a priori* defined as a separate replication process [6].

The computational nodes of the grid system can be characterized by a a *computing capacity vector* $CC_{batch} = [cc_1, \ldots, cc_m]$, where cc_i denotes the computing capacity of the node i. Each cc_i parameter ($i = 1, \ldots, m$) can be expressed by clock frequencies or by MIPS (Million Instructions Per Second) calculated for CPUs in the resources. The estimation of the prior load of each computational node from a given M_{batch} set is defined by a *ready times vector* $ready\_times_{(batch)} = [ready_1, \ldots, ready_m]$. The workload and computing capacity parameters for tasks and computing grid nodes can be generated by using the Gamma probability distributions for the expression of tasks and machines heterogeneities in the system (see [5] , chapter 2 , for details).

Data-aware task execution time model

We use the *Expected Time to Compute (ETC)* matrix model [1] for an estimation of times needed for the completion of the tasks assigned to the grid resources assuming also the data transmission times from the data nodes. A general concept of conventional ETC matrix model, used very often for solving the independent grid scheduling problems is based on the ETC array structure $ETC = [ETC[j][i]]_{n \times m}$, where $ETC[j][i]$ denotes an expected (estimated) time needed for the computing the task t_j at the resource m_i. The values of $ETC[i][j]$ parameters depend depend on the processing speed of the machines, to which they are assigned. However, in data-aware scheduling, the data transmissions times must be included into the model. Let us denote by $TT[i][j][f_{(p,j)}]$ a time needed for the transfer of the data file $f_{(p,j)}$ ($p \in \{1, \ldots, r\}$) from the data host $dh_{(p,j)} \in D_j$ to

the computational node m_i. This parameter can be calculated as follows [6]:

$$TT[i][j][f_{(p,j)}] = response_{time}(dh_{(p,j)}) + \\ + \frac{Size[f_{(p,j)}]}{B(dh_{(p,j)},i)} \quad (1)$$

where $response_{time}(dh_{(p,j)})$ denotes a time needed for receiving the first byte of the data file $f_{(p,j)}$ by the computational node m_i calculated from the moment of receiving data request by the data host $dh_{(p,j)}$, and $B(dh_{(p,j)}, i)$ denotes a bandwidth of the (logical) link between $dh_{(p,j)}$ and m_i.

The impact of the data transfer time on the task completion time depends on the mode, in which the data files are processed by the task. The are two main such scenarios which can be considered: (a) in the first scenario all data files needed for the execution of the task t_j are transferred *before* the computational process starts, and (b) the second scenario, where it is assumed that those data files which are not necessary for the initialization of the the execution of task t_j may be sent to the computational node later during the calculation process (the files are accessed as data streams during the calculations).

Let us denote by We denote by $completion[j][i]$ an estimated completion time for the task t_j on machine m_i, calculated from the task's submission till its completion in node m_i with the assumption of the access and transfer of all required data from the data hosts. In the first scenario this parameter can be calculated as follows:

$$completion[j][i] = \sum_{f_{(p,j)} \in F_j} TT[i][j][f_{(p,j)}] + ETC[j][i]. \quad (2)$$

where $\sum_{f_{(p,j)} \in F_j} TT[i][j][f_{(p,j)}]$ denotes the total time required for the 'sequential' transfer of all data files needed for the execution of task t_j.

In the second scenario (case(b)) the completion times for computational machines and tasks are calculated in the following way:

$$completion[j][i] = \max_{f_{(p,j)} \in \widehat{F_j}} TT[i][j][f_{(p,j)}] + \\ + \sum_{f_{(l,j)} \in [F_j \setminus \widehat{F_j}]} TT[i][j][f_{(l,j)}] ETC[j][i]. \quad (3)$$

where $\widehat{F_j}$ denotes a set of data files which are transferred prior the task execution. We will use the above $completion[i][j]$ parameters for the definition of the optimization criteria (schedulers' performance measures) in our simply empirical analysis presented in the next section.

For making the system easily adaptable to various scheduling scenarios, we consider the data hosts as the data storage centers separated from the computing resources. The scalability and effectiveness of the whole such system depends strongly on the replication mechanism and the resource data storage and computation capacities, which in some cases can be the main barrier in the schedulers' performance improvement. In our previous works [5,7] we assumed that each computing resource has its own data storage module. In such cases the internal data transfer times were low and we ignored them.

EMPIRICAL ANALYSIS

In this section we present the results of a simple empirical analysis of the performance of two implementations of GA-based energy-aware schedulers for static and dynamic versions of the data-aware independent batch scheduling problem in grid. We have developed a *Data-Sim-G Batch* simulator by a simple extension of our previously defined *Sim-G Batch* grid simulation toolkit (see [5]) by a data processing module The GA-based schedulers were evaluated on two benchmarks composed by a set of static and dynamic instances generated by the grid simulator.

Scheduling Objectives

Scheduling phases in the data-ware scheduling are similar to grid scheduling without data sets, and most of the conventional grid scheduling objectives, such as makespan and flowtime, can be easily adapted to the data-aware scheduling. For the scenario presented in in the following way:

- **Makespan:**

$$Makespan = \min_{Sched} \max_{m_i \in M_{batch}} completion[m_i] \quad (4)$$

where $completion[m_i]$ is computed as the sum of the completion times of tasks assigned to

machine m_i (see Eq. 3);

- **Flowtime:**
  - Flowtime for a machine i can be calculated as a workflow of the sequence of tasks on a given machine m_i, that is to say:

$$F[i] = ready_i + \sum_{j \in Sorted[i]} completion[j][i] \quad (5)$$

    where $Sorted[i]$ denotes a set tasks assigned to the machine m_i sorted in ascending order by the corresponding ETC values.
  - The cumulative flowtime in the whole system is defined as the sum of $F[i]$ parameters, that is:

$$F = \sum_{i \in M} F[i] \quad (6)$$

Both objectives are minimized. We consider hierarchical optimization process with makespan as the privileged (major) criterion. Flowtime is optimized with a constrain of not increasing the generated best makespan value. The wider list of the scheduling criteria in data grids can be found in [2].

Genetic-based data-aware schedulers

As a result of the wide assortment of constraints and different optimization criteria in the grid scheduling, meta-heuristic methods are the effective solutions for data intensive grid scheduling problems [10]. Genetic-based schedulers can easily explore the robustness of the search space and they can tackle various scheduling attributes.

For solving the data-aware independent batch scheduling problem, we have used in this paper two implementations of simple genetic grid schedulers, similar to the methodologies used in our previous works, where the big set of benchmarks and instances of the problem has been defined (see [5] for the summary of the results). These implementations, namely *GA* and *StGA* differ in the replacement mechanisms. The general frameworks of the schedulers are based on classical (μ+λ) evolutionary strategy (see e.g. [9]), adapted to the scheduling problem through the implementation of the following genetic operators:

- **Initialization method:** Randomly generated initial population;
- **Selection method:** Linear Ranking Selection;
- **Crossover operator:** Partially Mapped Crossover (PMX);
- **Mutation operator:** Rebalancing;
- **Replacement operators:** Elitist Generational (GA) and Struggle (StGA).

The detailed definition of these techniques can be found in [5].

Data-aware Batch grid Simulator - basic concept

The main concept of *Data Sim-G Batch* simulator is presented in Fig. 1.

Figure 1: General concept of *Data Sim-G Batch*

We have extended the *Sim-G Batch* grid toolkit defined in [5] by an implementation of additional data processing module responsible for generating (i) a set of data files, (ii) a set of data hosts, (iii) data transmission time matrix, (iv) response time vector, and (v) bandwith vector. All those data are considered

as basic characteristics of an instance of the problem and together with (vi) workload vector of tasks, (vii) computing capacity vector, (viii) prior load vector, and (ix) ETC matrix are passed on to the selected scheduler, which computes the schedule of the task assignments to the machines. Finally, the scheduler sends the schedules back to the simulator, which makes the allocation.

Key input parameters for simulator and schedulers

The performance of genetic-based schedulers analyzed in two types of grid environment: static and dynamic. In both cases four grid size scenarios: small (32 hosts/512 tasks), medium (64 hosts/1024 tasks), large (128 hosts/2048 tasks), and very large (256 hosts/4096 tasks). The schedulers' key parameters, including mutation and crossover probabilities, population size and stopping criteria (can be the maximal number of evolution steps or termination time criterion, are presented in Table 1.

Table 1: Shedulers' key parameters for static and dynamic benchmarks.

Parameter	GA	StGA
evolution steps	$20 * m$	
pop. size (pop_size)	$4 * (\log_2(m) - 1)$	
cross probab.	0.9	1.0
mutation probab.	0.2	
termin. time crit.	40 secs ($static$) / 75 secs ($dynamic$)	

The values of key parameters for the simulator for static and dynamic grid scenarios are presented in Table 2.

$N(*, **)$ denotes the Gaussian distribution. The detailed interpretation of all parameters is available in [5].

Each experiment was repeated 30 times under the same configuration of operators and parameters.

Results

The averaged makespan and flowtime values are presented in Tables 3 and 4.

It can be observed from the comparison of the result that the struggle replacement mechanism has rather crucial impact on the performance of the genetic scheduler. In all instances but three, calculated for both criteria in static and dynamic scenarios, *StGA* outperforms classical *GA* scheduler. The minimization of the flowtime, where *StGA* was the best in all instances, is in fact noticeable if we have into account that flowtime was considered a secondary (less important) objective in the optimization process. Both schedulers are rather stable in the optimization, which is confirmed by the low values of the $C.I.$ parameters. Finally, compare to the results achieved by similar implementations of the schedulers but in the case, where data transfer times are ignored (see [5], Chapter 4 for details), the values of makespan and flowtime have increased average by 10–25 %, which confirms the high importance of this criterion in data intensive scheduling.

CONCLUSIONS AND FUTURE WORK

In this paper we have addressed a general problem of data-aware scheduling problem of tasks submitted independently by the grid end-users. We assumed that for the completion of each task there are required some data files distributed also in the grid system and stored at heterogeneous data hosts. We have formalized the transmission time, in a way that it can be easily integrated into classical optimization objectives of grid scheduling, namely makespan and flowtime expressed in the terms of completion times of task on computational grid nodes, where data can be transferred *a priori* or immediately during the task computation. For the empirical analysis, we have implemented two versions of simple genetic-based grid scheduler for solving the considered scheduling problem aiming to minimize both makespan and flowtime scheduling objectives in the hierarchical mode, with makespan as major (privileged) objective. The empirical analysis has been performed by using the developed *Data-Sim-G Batch* grid simulator.

Table 2: Parameter setting for the grid simulator static instances

	Small	**Medium**	**Large**	**Very Large**
Static Instances				
Number of hosts	32	64	128	256
Resource capacities (in MIPS)	$N(1000, 175)$			
Total number of tasks	512	1024	2048	4096
Workload of tasks	$N(250000000, 43750000)$			
Dynamic Instances				
Init. hosts	32	64	128	256
Max. hosts	37	70	135	264
Min. hosts	27	58	121	248
Add host	$N(625000, 93750)$	$N(562500, 84375)$	$N(500000, 75000)$	$N(437500, 65625)$
Delete host	$N(625000, 93750)$			
Total tasks	512	1024	2048	4096
Init. tasks	384	768	1536	3072
Workload	$N(250000000, 43750000)$			

Table 3: Average Makespan and Flowtime values (± %C.I.) for static instances (C.I.: confidence interval)

Scheduler	**Small**	**Medium**	**Large**	**Very Large**
Makespan values (in arbitrary time units)				
GA	**4171630.27**	4286741.95	4306153.30	4338090.10
	(±)0.5714%	(±)0.7950%	(±)0.9351%	(±)1.1843%
StGA	4072614.96	**4179528.76**	**4286350.17**	**4299442.95**
	(±)0.6421%	(±)0.7714%	(±)1.1750%	(±)1.5132%
Flowtime values (in arbitrary time units)				
GA	1213553487.5	2344982276.8	4427950665.1	8421751474.6
	(±)0.9532%	(±)0.7980%	(±)0.9792%	(±)0.9917%
StGA	**1205329495.4**	**2293768328.5**	**4401468978.4**	**8399042744.8**
	(±)0.9421%	(±)0.8765%	(±)1.3298%	(±)1.2276%

Table 4: Average Makespan and Flowtime values (± % C.I.) for dynamic instances (C.I.: confidence interval)

Scheduler	**Small**	**Medium**	**Large**	**Very Large**
Makespan values (in arbitrary time units)				
GA	**4148152.90**	**4188204.13**	4415066.05	4441820.13
	(±)0.7560%	(±)0.8501%	(±)1.0724%	(±)1.7805%
StGA	4262331.15	4199261.61	**4381408.54**	**4378104.73**
	(±)0.8109%	(±)1.4350%	(±)1.9363%	(±)1.8390%
Flowtime values (in arbitrary time units)				
GA	1286071183.744	2269852393.768	4536176645.169	8993540827.579
	(±)0.8102%	(±)0.9240%	(±)1.2912%	(±)1.7805%
StGA	**1244873655.6**	**2255039877.2**	**4498453672.8**	**8953201900.1**
	(±)0.8225%	(±)0.8905%	(±)1.3773%	(±)1.9150%

The results show that both GAs are effective methods for keeping the makespan and flowtime on rather low levels, although Struggle GA performed best.

In our future work, we would like to extend our empirical analysis for a wider class of the schedulers and scheduling criteria, and use similar concept for solving the cloud scheduling problems.

REFERENCES

[1] Ali, S., Siegel, H.J., Maheswaran, M., and Hensgen, D.: "Task execution time modeling for heterogeneous computing systems", *Proceedings of Heterogeneous Computing Workshop*, pp. 185–199, 2000

[2] Buyya, R., Murshed, M., Abramson, D., and Venugopal, S.: "Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost-time optimization algorithm", *Softw. Pract. Exper.*, Vol. 35(5), (2005), pp. 491–512.

[3] Chen D., Wang L., Wu X., Chen J., Khan S.U, Kolodziej J., Tian M., and Huang F.: "Hybrid Modelling and Simulation of Huge Crowd over a Hierarchical Grid Architecture", *Future Generation Computer Systems*, DOI: 10.1016/j.future.2012.03.006, 2013.

[4] Kliazovich, D., Bouvry, P., Audzevich, Y., and Khan, S.U.: "GreenCloud: A Packet-level Simulator of Energy-aware Cloud Computing Data Centers", in *Proc. of the 53rd Globecom*, Miami, FL, USA, December 2010.

[5] Kołodziej J.: *Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems*, in *Studies in Computational Intelligence* Springer series, Vol. 419, Springer Vlg., Berlin-Heidelberg, 2012.

[6] Kołodziej, J. and Khan, S. U.:"Data Scheduling in Data Grids and Data Centers: A Short Taxonomy of Problems and Intelligent Resolution Techniques", *Transactions on CCI*, Vol X, LNCS 7776, pp. 104–121, 2013.

[7] Kołodziej, J. and Khan, S. U.: "Multi-level Hierarchical Genetic-based Scheduling of Independent Jobs in Dynamic Heterogeneous Grid Environment", *Information Science*, Vol. 214(2012), pp. 1–19, 2012.

[8] Kosar, T. and Balman, M.: "A new paradigm: Data-aware scheduling in grid computing", *Future Gener. Comput. Syst.*, Vol. 25(4), (2009), pp. 406–413.

[9] Michalewicz, Z.: *"Genetic Algorithms + Data Structures = Evolution Program"*, Springer, 1992.

[10] Venugopal, S., and Buyya, R.:"An SCP-based heuristic approach for scheduling distributed data-intensive applications on global grids", *J. Parallel Distrib. Comput.*, vol. 68, pp. 471–487, 2008.

[11] Wang L, and Khan, S.U.: "Review of Performance Metrics for Green Data Centers: A Taxonomy Stud", *Journal of Supercomputing*, pp. 1–18, 2011.