Dynamic Scheduling for Batch Data Processing in Parallel Systems

Mojahid Saeed Osman, Malick Ndiaye and Abdulrahim Shamayleh
Systems Engineering Department, King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia

Keywords: Data Processing, Dynamic Scheduling, Batch Processing.

Abstract: In many operations time factor is a main constraint. Being able to optimally execute them while minimizing the amount of resources used remain a challenge in many business areas. In this paper we propose a model to optimally schedule processing a set of tasks which are part of network of operations on processors that are all capable of handling these tasks. The objective is to process these tasks and operations within a cut-off time while satisfying all the precedence constraints using the minimum number of resources, i.e. processors.

1 INTRODUCTION

Batch processing and its applications are critical in most organizations because many common business processes are amenable to batch processing. It is a popular problem in both manufacturing and service industries with a wide range of application such as chemical plants, virus scanning, banks, etc.

The focus of this work is on batch processing optimization in financial institutions where end of day activities and data such as updating information at the end of the day, generating reports, printing documents, and other non-interactive tasks must be completed reliably within a certain business deadlines. Usually processing start after business hours at night for a certain number of hours till the deadline is reached.

Several benefits can be achieved from batch processing which include shifting the time of the job processing to when the computing resources are less busy, it allows the system to use different priorities for batch and interactive work, and sharing of computer resources among many users and programs which will translate into keeping high overall rate of utilization for resources especially the expensive ones.

The optimization model developed in this paper attempts to optimally schedule processing a set of tasks which are part of network of operations on processors that are all capable of handling these tasks. The objective is to process these tasks and operations within a cut-off time while satisfying all the precedence constraints using the minimum number of resources, i.e. processors.

2 LITERATURE SURVEY

In many operations time factor is a main constraint. Being able to optimally execute them while minimizing the amount of resources used remain a challenge in many business areas. Extended literature is available about various strategies on scheduling and processing jobs using parallel systems. These strategies usually depend on the areas of applications.

In the context of Data Processing, conventional job scheduling strategies, e.g. FCFS (First Come First Served), Longest Job First (LJF) Algorithm, Backfilling, etc., have been studied and their performance assessed. One can find detailed classification and a review of optimization methods in (Mendez et al., 2006). In (Aida, 2000) the author discusses scheduling performance based on the job sizes. Most of the algorithms performances are not sensitive to job sizes while some like the LJF might be affected.

Beside the conventional approaches, some authors have introduced heuristic techniques to manage the allocation of jobs to servers. In (Page et al, 2010), a Genetic Algorithm are combined with other heuristic algorithms is efficiently used to dynamically schedule the tasks minimizing potential idleness in the context of heterogeneous distrusted systems. A wide review of heuristics and meta-heuristics methods for Grid Scheduling problems is presented in (Xhafa and Abraham, 2010). Different scheduling criteria are presented to discuss the complexity and difficulty of achieving efficient Grid Schedulers.

Other category of studies attempt to minimize the total makespan without a particular focus on the allocation rule of jobs to servers. They used different criteria such as total completion time, total weighted completion time and makespan, in heuristics solutions approaches. In (Damodaran and Vélez-Gallego, 2012), they compare the performance of a Simulated Annealing (SA) approach, a Modified Delay (MD) heuristic and a Greedy Randomized Adaptive Search Procedure (GRASP) in minimizing the makespan of parallel batch processing machines. In (Lim and Cho, 2007), the authors propose a CPU process scheduling algorithm using fuzzy inference with user models. They classify tasks into three categories, batch, interactive and real-time processes, and models user's preferences to each process class. The algorithm assigns the scheduling priority of each process according to the class of the process and user's preference through the fuzzy inference. Another important area of application is in database management systems when queries are to be addressed. Single query optimization has been well studied with various strategies depending on the architecture of the database system, (Mehta et al., 1993). However the use of batch queries scheduling in multi-user environment is still to be properly tackled.

3 BATCH DATA PROCESSES SCHEDULING PROBLEM

The problem deals with the scheduling and allocation of a set of data files to a set of processors for maximizing some performance measures while satisfying all the priority and precedence constraints.

Data files cannot pre-assign to processors, however, only data files that are identified available for IC processing are to be scheduled and allocated to processors based on the precedence weight and predetermined scheduling priority. The processing time of a data file is a function of file size/number of records, and sizes of data files change after performing each IC processing task; it can be assumed that the size of a data file changes proportionally with the original file size.

Each file may require performing a number of computing instruction (CI) processing and I/O reading. CI processing and I/O readings can be executed in parallel operating systems. CI processing tasks can split into two or more processors. CI processing tasks can be processed in processors until it get interrupted by an I/O reading task or reaches “end of processing”.

In the processor set, there are several processors running concurrently (multiprocessing), each processor is assigned to ONLY one CI processing task at a time; we cannot perform more than one task at a time in a single processor. The reservation or pre-allocation processor is banned.

The notion of interruption must be taken into account. It is essential to the functioning of the operating system: a data file A is allocated to a processor to perform CI processing task, if the program reaches a reading input-output (I/O) task, then the data file A is interrupted in order to allow the processor to run another data file that was waiting, say B for instance. At the end of the I/O instruction task, data file A will return in the queue in a position which depends on an updated precedence weight and a predefined scheduling priority (dispatching priority DP) and expect to benefit again from the processor when its turn comes. As a result, a task has a total duration that varies according to the number of concurrent tasks and multiprocessing – we use these rules to set priorities among the data files.

4 PROPOSED APPROACH – BATCH DATA PROCESSES SCHEDULING (BDPS) ALGORITHM

We present a dynamic optimization framework for the batch data processes scheduling based on a dynamic algorithm. The batch data processes scheduling (BDPS) algorithm is an iterative process optimizing the allocation several processors to different tasks and scheduling batch data processes.

The BDPS uses an iterative approach, Step 1 reflects a preparatory stage where the initial data is set up and BDPS algorithm begins, then BDPS enters a loop that includes repetitive steps. The first task in this loop, Step 2, is to update and increment the iteration clock by one time unit (estimated by the time it takes to process the smallest size unit of data file). At each time T the BDPS algorithm considers allocating data files available for IC processing to several processors so Step 3 involves setting up data files subset; only data files that are not immediately preceded by other data files can be IC processed at current time T and, therefore, belong to the data file subset (I'). In step 3 we also set the data file weight based precedence/dependency matrix. Next, in Step 4 BDPS solves an integer network optimization model to determine the allocation of data file, that are available for IC processing at current time *T*, to different processors. This optimization model requires input parameters that are predetermined using the precedence matrix and scheduling priority. In Step 5, BDPS updates the availability of files and processors. Then in Step 6 the termination condition is checked and the algorithm continues to repeat Steps 2-6 as long as there are data files left in queue using one time unit increment until the termination condition is satisfied. The steps in the BDPS algorithm are defined formally next after the notation is presented. Prior to presenting the steps of BDPS algorithm in greater detail, we introduce the notation shown in Table 1.

Table 1: BDPS notations.

	Description
I	is the set of all data files at any time *t*
I'	is a subset of data file that are available for CI processing at any time *t*
K	is the set of all processors
K	is a subset of processors to be allocated at any discrete time *t*
f^t_i	is equal to 1 if data file *i* is available for CI processing at discrete time *t*, and 0 otherwise
l_{ij}	is equal to 1 if data file *j* **immediately** precedes data file *i*, and 0 otherwise (*parameters of precedence/dependency matrix*).
q_i	is equal to the number of times data file *i* been CI processed; is incremented by 1 every time data file *i* being CI processed
n_i	is equal to the number IC processing tasks required for data file *i*
p_k	is equal 1 if processor/server *k* is available to receive data file, and 0 otherwise.
T	Clock discrete time
x_{ik}	is 1 if data file *i* allocated to processor *k*, and 0 otherwise
u_i	The total CPU time required to process data file *i*
α_i	The data file weight based *precedence/dependency matrix*
β_i	The data file scheduling priority
e_i	The multiprocessing of data file *i*
s_i	The data file size at T=0

The main steps of the BDPS algorithm are detailed as follows.

Given: l_{ij} , n_i $\quad\forall\ i$ and $j(\ i\neq j)$

Step 1*:* Initialization

– Set $q_i = 0,\ p_k = 1$ $\quad\forall\ i\in I,\ k\in K$

Step 2: Update clock

– Set $T = 0$

Step 3: Setup I' subset

– Set $f_i^T = 1$ if $\sum_{j=1}^{J} l_{ij}^T = 1\ \ \forall\ i$

– if $f_i^T = 1$ then set $i\in I'$

– Update data file weight

Set $\alpha_i = \sum_{j=1}^{J} l_{ij} \qquad \forall\ i$

Step 4: Solve the optimization model to allocate data files to processors

Objective Function:

$$\text{- } Max \sum_{i\in I'}^{I'}\sum_{k=1}^{K} \alpha_i\beta_i x_{ik} \tag{1}$$

Subject to:

$$\text{- } \sum_{k=1}^{K} e_i x_{ik} \le M f_i^T \ \ \forall\ \ i\in I'\ \ and\ M\le|K| \tag{2}$$

$$\text{- } \sum_{i\in I'}^{I'} x_{ik} \le p_k^T \qquad \forall\ \ k\in K \tag{3}$$

$$\text{- } \sum_{k=1}^{K} \alpha_i\beta_i x_{ik} \le \sum_{k=1}^{K} \alpha_j\beta_j x_{jk} \quad \forall\ \ i,j\in I'\ and\ i\neq j \tag{4}$$

$$\text{- } x_{ik} = 0 \quad or \quad 1 \quad \forall\ \ i\in I\ and\ k\in K \tag{5}$$

The objective function (1) maximizes the performance measures, sum of importance weight and scheduling priority of individual data files. Constraint (2) ensures that a data file *i* must be available for allocation to a processor. Constraint (3) ensures that exactly one data file is allocated to a single processor. Constraint (4) imposes the precedence rule. Constraint (5) declares that the decision variable x_{ik} is binary.

Step 5: Update f_i^T Matrix

– If $x_{ik} = 1$ then $f_i^T = 0,\ \ p_k^T = 0,\ \ p_k^{T+\Delta} = 1,\ and$

$q_i = q_i + 1 \qquad\qquad \forall\ \ i\in I',\ k\in K$

– If $q_i = n_i$ then $l_{ij}^{T+\Delta} = 0\ \ \forall\ \ j\ else$

$f_i^{T+\Delta} = 1 \qquad \forall\ \ i\ \&\ j\in I',\ i\neq j$

Step 6: Check termination condition

If $\sum_{i=1}^{I} q_i = \sum_{i=1}^{I} n_i$ then **Stop** else Go to **Step 2**

5 ILLUSTRATIVE EXAMPLE

We illustrate our BDPS approach by solving the batch data processes scheduling problem shown in Figure 1. In this problem, 6 data files are to be scheduled for processing in 2 available processors.

The initial precedence matrix, and data file weights and sizes at T=0 are given in Tables 2 and 3, while the required IC processing times, scheduling priority and multiprocessing of each data file *i* are shown in Table 4.

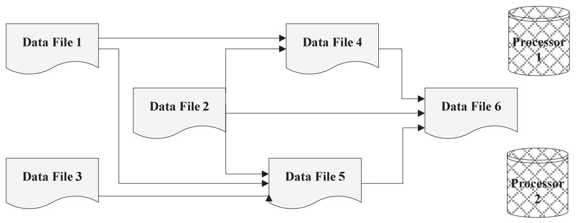

Figure 1: Illustrative example problem representation.

Table 2: Precedence Matrix for l_{ij} *at T=0.*

	j					
i	**1**	**2**	**3**	**4**	**5**	**6**
1	1	-	-	-	-	-
2	-	1	-	-	-	-
3	-	-	1	-	-	-
4	1	1	-	1	-	-
5	1	1	1	-	1	-
6	-	1	-	1	1	1

Table 3: Data File Initial Weight and Size *at T=0.*

	i					
	1	**2**	**3**	**4**	**5**	**6**
α_i	1	1	1	3	4	4
s_i	2	2	2	2	2	2

Table 4: Required IC processing times, Scheduling priority and Multiprocessing of Data File *i*.

	i					
	1	**2**	**3**	**4**	**5**	**6**
n_i	1	2	1	1	1	1
β_i	5	1	1	1	1	1
e_i	1	2	1	1	1	1

For each discrete time *T*, we summarize the data file subsets *(I′),* weights (α_i) based precedence matrix, and availability (f_i) for CI processing in Table 5. In Table we show the availability of processors (p_k) for CI processing at discrete time *T*.

Table 5: Data File Weights (α_i), subset *(I′)* and Availability (f_i) for CI processing.

	i					
T	**1**	**2**	**3**	**4**	**5**	**6**
0	3|✓	4|✓	2|✓	2	2	1
1		4|✓	2|✓	2	2	1
2		4|✓	2|✓	2	2	1
3		4|✓	2|✓	2	2	1
4			2|✓	2|✓	2	1
5					2|✓	1
6					2|✓	1
7						1|✓
8						1|✓

Files with check marks (✓) are available for IC processing at that time and, therefore, belong to data file subsets (I′)

The optimal schedule obtained as results of solving the integer optimization model are reported in Table 6. Each data file *i* is allocated to processor *k* precisely at time *T*.

Table 6: BDPS solution for illustrative example.

	i					
T	**1**	**2**	**3**	**4**	**5**	**6**
0	*k:1*	*k:2*				
1		*k:2*				
2		*k:1&2*				
3		*k:1&2*				
4			*k:1*	*k:2*		
5	******** Processors are not available*******					
6					*k:1*	
7	******** Processors are not available*******					
8						*k:1*

The above illustrative example was solved using GAMS 22.6 using the CPLEX solver. The system used to solve the BDPS algorithm is a HP PC with 32-bit Windows 7 and Intel(R) Core(TM)2 Duo CPU 2.80 GHz processor, 4GB of RAM, and a 150GB hard drive. The allocation and dynamic schedule obtained as final solution reflect the accuracy of the BDPS model formulations. BDPS model solved the illustrative example in approximately 25 CPU milliseconds. It is anticipated that even if for too large number of data files, BDPS can still yield optimal solutions in reasonable computer CPU time and memory.

6 CONCLUSIONS

In this paper, we proposed a dynamic scheduling algorithm for batch data processing (BDPS) based on a hybrid approach, heuristic approach with optimization model. The objective is to schedule and allocate data files with different sizes, importance weights, and priorities to the minimum number of processors for maximizing some performance measures while satisfying all the precedence constraints.

The BDPS is a novel approach provides competitive solutions; the results reveal the effectiveness of the proposed BDPS approach for solving dynamic scheduling problem. The scheduling for batch data in parallel systems is only one example of a problem that can be modeled as a dynamic scheduling problem with scheduling priority and precedence constraints; it is anticipated that other problems of this type can benefit from the proposed BDPS algorithm. This effort has the potential to lead to more improvements to the dynamic scheduling.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the support provided by King Fahd University of Petroleum & Minerals (KFUPM).

REFERENCES

Aida K., 2000. Effect of job size characteristics on Job scheduling performance. *Lecture Notes in Computer Science* Volume 1911, pp 1-17

Damodaran P., Vélez-Gallego M., 2012. A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times. *Expert Systems with Applications 39:1451-1458.*

Lim S., Cho S., 2007. Intelligent os process scheduling using fuzzy inference with user models. *Lecture Notes in Computer Science* 4570:725-734

Mehta M., Soloviev V., DeWitt D., 1993. Batch scheduling in parallel database systems. *9th International Conference on Data Engineering* 400-410

Mendez C., Cerda J., Grossmann I., 2006. State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Computers and Chemical Engineering 30:913-946*

Page A., Keane T., Naughton T., 2010. Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system. *Journal of Parallel and Distributed Computing 70:758-766*

Xhafa F., Abraham A., 2010. Computational models and heuristic methods for Grid scheduling problems. *Future Generation Computer Systems 26:608-621.*