Batch systems: Optimal scheduling and processor optimization

Rushi Agrawal*†‡ and Vaishali Sadaphal‡
*Student Author
†Indian Institute of Technology Hyderabad, Hyderabad, India
‡Tata Research Development and Design Center, Pune, India
Email: rushi@iith.ac.in, vaishali.sadaphal@tcs.com

***Abstract*—In this paper, we define various problems encountered in batch processing in modern enterprise systems. We focus our attention on scheduling of batch jobs on multi-processor systems such that the batch finishes in minimum time with minimum resource utilization. The contribution of this paper is two folds: (1) study properties of dependency graph of batch jobs to identify that for fat graphs the state-of-the-art CP/MISF (critical path/most immediate successors first)[1] scheduling algorithm computes near optimal schedules; (2) further, we propose an efficient graph pre-processing method to shorten the time taken to compute schedule of batches that have dense dependency graphs. We show that an overall gain of 0 - 50% in the execution time is achieved by pre-processing.**

I. Introduction

Data centers typically cater to two types of workload: transactional and batch. The transactional workload is processed in real-time and needs to meet per-request service level objectives (SLOs) defined on metrics such as response time. The batch workload, on the other hand, consists of jobs such as data consolidation, data compliance checks, maintenance jobs, etc. The desired performance of a batch workload is defined by metrics such as the batch completion time (time within which all batch jobs should be completed) and peak resource requirements. The past literature contains a lot of work on performance and capacity analysis of transactional systems [2], [3]. Analysis of batch workloads, though equally important, has received relatively lesser attention. In this paper, we focus on analysis of batch processing systems.

Both transactional and batch systems need answers to similar questions such as: what is the cause of delay in completing a transaction or a batch?, what is the capacity of the system - How much resource is available?, how much more workload can be processed in the available resource?, etc. Though there are commonalities between performance and capacity analysis of transactional and batch systems, the problem of scheduling of jobs is specific to batch systems.

A typical batch system is characterized by: (1) A set of jobs: each job is characterized by its execution time. (2) Precedence or dependency relationships: a job can not start unless and until each and every job on which it is dependent is completed. The dependency relationship is represented as a directed acyclic graph. (3) Batch completion time constraint: batch execution should finish before batch finish time, $t_{bc} \leq T_{bc}$, where t_{bc} is the time required to complete the batch and T_{bc} is maximum time before which the batch must finish. (4) Resource constraint: there is a threshold on the maximum amount of resources that could be used at a point of time. In other words, $p_{res} \leq P_{res}$ where p_{res} is the peak amount of resource used at any point of time and P_{res} is the maximum threshold beyond which the resource utilization should not exceed. Any additional use of resource beyond the threshold results in additional financial charges.

Scheduling jobs to meet the batch completion time constraint and resource utilization constraint simultaneously is a challenging task. Work has been done in the past on scheduling of multiprocessor systems and batch systems. [1],[4] addresses scheduling of processes on the multiprocessor systems. Work has been done in the area of multiprocessor systems where the processing speeds of processors is not equal [5]. [6] explores using frequently occurring subgraphs for computing schedules.

In Section II, we discuss in detail various versions of scheduling problems related to batch system. We also identify contribution of the paper in this section.

II. Scheduling of Batch Systems

The batch systems run on different types of infrastructures, viz. multiprocessor systems or mainframes. In multiprocessor systems, each processor may provide equal fixed CPU cycles/sec, or processors may provide unequal CPU cycles/sec. The execution time of each job is dependent upon how much CPU cycles/sec are provided to it. In case of multi-processor system with equal and fixed CPU cycles/sec the execution time of job is independent of the processor on which it is scheduled to run. This makes scheduling simpler than in the case of multi-processor system with unequal CPU cycles/sec. In case of multiprocessor systems with unequal CPU cycles/sec, the execution time of a job depends upon the processor on which it is scheduled to run. In this case the possibilities of scheduling jobs multiply exponentially. In mainframes, there is a provisioning of maximum CPU cycles or MIPS (millions of instructions per second) that could be provided at a point of time. Mainframes are also responsible for introducing the concept of LPARs: Logical PARtitions on the processing unit which provides separation and different speeds for processes to execute on them.

In case of each of the above infrastructures, different issues need to be addressed while scheduling jobs. In the following section we discuss various problems related to scheduling in batch systems.

A. Multiprocessor environment: Minimize the batch completion time given fixed number of processors

The most common requirement of batch systems is that the batch is required to be scheduled such that it satisfies the batch completion time constraint within the available resource. This is problem can be formulated as an optimization problem: Given, (1) a set of n jobs, $j_1, j_2, \ldots, j_n$, (2) execution times of jobs, $t_1, t_2, \ldots, t_n$, (3) directed acyclic graph as dependency relationships, $G(V, E)$ such that $V = \{j_1, j_2, \ldots, j_n\}$ and E specifies dependence between jobs. (4) a fixed number of identical processors, P, minimize the batch completion time, t_{bc}.

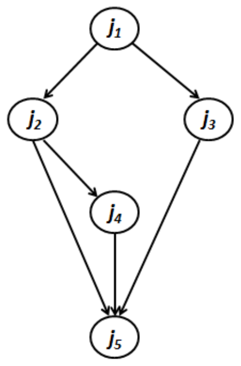

Fig. 1. A directed acyclic dependency graph with a redundant edge between j_1 and j_3.

The edges in the dependency graph signifies the partial ordering of the graph. In simpler words, a job cannot be scheduled before all of its immediate predecessors have finished execution. In Figure 1, j_5 cannot be scheduled before j_2, j_4 and j_3 finish execution. This problem is an NP-Hard problem [7].

B. Incremental computation of schedules

The batches in modern enterprise systems change every day. The workload of the jobs changes. The dependency relations between the jobs change. However, it has been noticed that the change in the dependency relationship may not be very large. Subgraphs in the dependency graph may repeat. The execution time of only a few of the jobs may change and for most of the jobs it may remain the same [6].

In case of small scale changes, it should be possible to compute schedules incrementally without processing the complete dependency graph. Incremental computation of schedules or computing schedules only for a subgraph needs to be explored. It should be possible to answer following questions efficiently without requiring computation of overall schedule: Is it possible to re-compute schedule efficiently if (1) workload of only a few nodes change, (2) workload of nodes of a subgraph changes, (3) a node is added/deleted to/from the batch system, (4) a subgraph of nodes is added/deleted. There is a need to explore efficient methods to answer the above problems. Approaches which involve analyzing past history of graphs, analyzing frequently occurring subgraphs, pre-computation of their schedules may be used to possibly pre-compute schedules as well. We propose to address this in future.

C. Mainframe environment

In case of mainframes, a pool of CPU cycles/sec with threshold on maximum CPU cycles that can be used at a point of time is given. Batches may run on different LPARs. And each LPAR may have different maximum MIPS allocated to it. A batch can be scheduled on an LPAR with higher or lower speed of execution in order to minimize the batch completion time. Separate CPU cycle provisioning to LPARs, flexibility of assigning jobs to LPARs with different speeds, and meeting batch completion time and resource constraint throws various challenges in scheduling jobs optimally on mainframes. We propose to address these problems in the future.

III. Scheduling in multi-processor systems

In this section, we look at the problem defined in Section II-A in greater detail. At the end of the day, the operators have an understanding of jobs in their batch and their dependencies. As a result, full schedule can be computed before actually assigning (possibly unfitting) jobs to processors. As the problem is strongly NP-hard, computation of optimal schedule is infeasible.

CP/MISF algorithm, the best algorithm known so far, generates priority of jobs statically, based on the 'maximum execution path length': the costliest path to the end node in the dependency graph with respect to time. For example, for job a with execution time e_a, if there are only two paths to the end node z in the dependency graph: paths $a{\rightarrow}b{\rightarrow}c{\rightarrow}z$ and $a{\rightarrow}d{\rightarrow}z$, with nodes b, c, d, having execution times e_b, e_c and e_d respectively, the longest path execution time will be maximum of (e_b+e_c) and (e_d).

A. Approach

The calculation of 'maximum execution path length' for each node in the precedence graph makes the complexity of CP/MISF algorithm proportional to E^2 where E is the number of edges in the graph. This makes the execution time consuming for graphs with large number of edges. On some of the precedence graph of more than 1000 nodes from batch systems of data centers of a financial bank, CP/MISF algorithm took almost two hours to calculate schedule. In such cases, we propose to pre-process graphs to compute schedules more efficiently. We propose that *dense* graphs be pre-processed by removing the 'redundant edges' to reduce the execution time of CP/MISF algorithm. We show that overall gain of 0-50% in the execution time is achieved.

B. Redundant edge removal

Batch graphs created from enterprise batch data usually contain hundreds of nodes with thousands of edges. Of these edges, a significant number of edges contain redundant information. In the graph 1, the edge $j_2 \rightarrow j_5$ is redundant as the information portrayed by this edge is already contained in other two edges (j_5 is to be scheduled after j_4 already implies that j_5 is scheduled after j_2 as j_4 is to be scheduled after j_2). Refer figure 1.

We propose an algorithm to remove all the redundant edges from a batch graph. We measure the performance of our algorithm on the collection of task graphs given in Standard Task Graph [8], and present the results in the next section. The proposed algorithm first selects edges which are potential candidates for redundancy, and then checks whether the selected edge can actually be harmlessly removed.

Algorithm: For a batch graph G of n nodes, let j_1, j_2,... j_n represent the nodes of the graph. We first calculate the topological ordering of nodes and save the node ids in the topological order in a list `T`. Also, along with finding out the order of topological sort, we calculate the 'level' of each node in the graph. A level of a node is the longest path from the start node to the current node. We represent level of a node i by lev_i. The $in\_neighbour()$ function returns list of all the immediate predecessors of the argument node.

```
for $i$ in $T$ do
  for $j$ in $in\_neighbour(i)$ do
    if $lev_j < lev_{i-1}$ then
      $remove\_if\_redundant(i, j)$
    end if
  end for
end for
```

The $remove\_if\_redundant(a, b)$ function essentially implements breadth first search algorithm: disregarding the edge j$\rightarrow$i, the breadth-first search starts from node i $in\ reverse$. The search stops when node j is found, or all the nodes found so far have level greater than or equal to lev_j

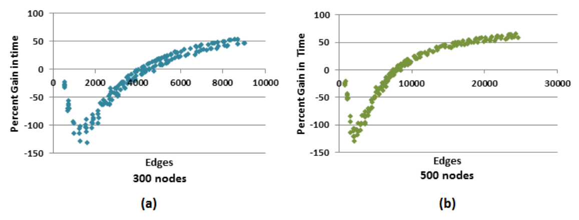

Fig. 2. Effect of number of edges on gain in execution time: (a) 300 nodes, (b) 500 nodes.

IV. Experiments and Results

A. Dependence of deviation of schedules on graph parameters

Although CP/MISF gives the results very close to the optimal value in a comparatively very small time, it is worth studying the pattern - if any - of how much the algorithm deviates from the optimal with respect to the change in the graph properties.

We generated directed acyclic graphs based on the following parameters: (1) fatness, (2) density and (3) number of nodes[9]. Fatness of the graph is defined as the inverse of the number of levels in the graph; density of a graph is the ratio of actual number of edges in the graph to the maximum number of edges that are possible for the graph with same number of nodes. The default values of parameters are: (1) fatness, f: 0.5 (2) density, d: 0.5 (3) number of nodes/jobs: 100.

For all these graphs, we calculated schedule lengths with two identical processors. We calculated by how much amount (in percentage difference from the optimal) the schedule generated by CP/MISF algorithm deviate from the optimal value. As we generated thousands of graphs, it was not feasible to calculate optimal schedules for all the graphs, so we used lower bound [10] as the optimal value for each of these graphs. The lower bound gives results very close to the optimal value for graphs with few hundred of nodes. For all 180 graphs of Standard Task Graph [8] of each of 50, 100, 300, and 500 nodes, for which optimal schedule was provided on [8], the lower bound gave values equal to the optimal schedule. In order to make the data appear more meaningful, for each set of parameters, we generated 10 graphs, and averaged the value of the deviation from optimal schedule.

Effect of density of the graph: Figure 3(c) shows a plot of density vs. deviation from optimal or error, e for fatness 0.4, 0.5, and 0.6. It can be observed that the error increases with increasing density of the graph. However, as the fatness increases, the error reduces. The reason for this trend can be explained in the following manner: for fixed fatness, the number of levels in a graph are same; but increase in density will introduce more number of edges in the graph and therefore increases the number of paths (or in other words, the number of possible schedules) in the graph. As CP/MISF algorithm tries to select a schedule very close to optimal, the increase in density increases the number of schedules to pick from resulting in the decrease in the accuracy of the algorithm.

Effect of fatness of the graph: From figure 3(b), it can be observed that when the fatness of the graph is small and large the error is small. When the fatness is very low, the graphs tends towards a linear chain of nodes. As the number of edges in the graphs for same fatness is nearly the same, such a form of graph has relatively less number of possible schedules, thereby giving a very less chance for the algorithm to select a schedule very far from optimal value. When the number of fatness is very high, following reason can be attributed to the behaviour: at any instant in the execution of schedule generation, the number of available jobs is very high as compared to the number of idle ('schedulable') processors. As a result, the generated schedule has a processor idle for an extremely small fraction of the total execution time. The optimal schedule is no different. This means that there are a large number of schedules, but all those schedules also lie very close to the optimal value, which results in the depicted trend.

Effect of size of the graph: The plot in figure 3(a) depicts the error in the schedule decreases as the size of the graph increases. The percentage error in the schedule is the actual error in time units, divided by the length of the optimal schedule. As the number of nodes increase keeping all the other parameters same, the optimal schedule increases linearly; so, although the actual deviation from optimal is still of the same order, the large denominator makes the percentage deviation appear small. However, even in case of larger graphs, if the fatness > 0.5, the error is small for the reasons mentioned in the above section.

Thus, we conclude that the CP/MISF algorithm gives near optimal results for fat graphs.

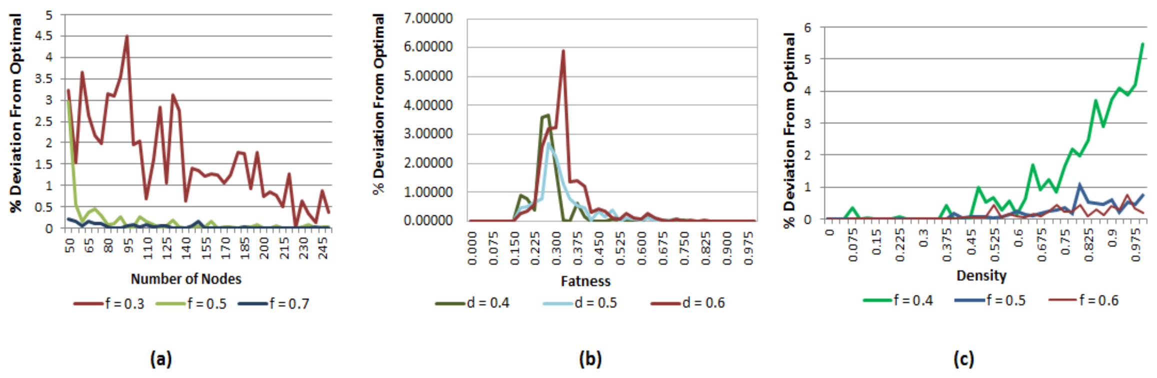

Fig. 3. Effect of (a) number of nodes, (b) fatness, (c) density on the accuracy of schedule.

B. *Benefit in execution time by pre-processing the graph*

We implemented and ran the algorithm for all of the 300-node 180 graphs of Kasahara's Standard Task Graph dataset [8]. As we found out, the graphs from which more number of edges were removed gave up to 50% increase in total execution time. A clear relationship between the number of edges in the original graph and the computational speed-up can be seen from the graph Figure 2 (a),(b).

Figure 2 shows the plot of number of edges and gain in execution time in percentage by pre-processing the graph of size (a) 300 nodes and (b) 500 nodes. It can be observed that the gain in execution time is positive for graphs with ratio of number of nodes to number of levels in the graph greater than 13 for a graphs of 300 jobs, and greater than 15 for graphs with 500 jobs. We propose to take a decision of pre-processing the graphs based on this metric: ratio of number of nodes to the number of levels in the graph.

V. Conclusion and Future Work

In this paper, we addressed various problems in batch systems. We addressed problem of scheduling and optimizing the resources in detail. We address the problem of scheduling of batch jobs on multi-processor systems. Our contributions is three folds: (1) We study the graphs properties of the dependency graph and observe that for the fat graphs the state-of-the-art algorithm (CP/MISF [1]) gives near optimal results and there is no need to run the computationally intensive optimal algorithm. (2) Further, we propose an efficient method of computing schedules of batches that have dense dependency graphs. (3) We propose that the graphs be pre-processed to reduce the execution time of CP/MISF algorithm. We show that overall gain of 0-50% in the execution time is achieved by pre-processing.

In future, we plan to address better methods of computing schedules using dynamic information. We also propose to address computing schedules incrementally when a portion of the dependence graph changes. We also plan to address scheduling problem and MIPS optimization in mainframe environment.

References

[1] H. Kasahara and S. Narita, "Practical multiprocessor scheduling algorithms for efficient parallel processing," in *IEEE Transactions on Computers*, 1984.

[2] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang, "Towards highly reliable enterprise network services via inference of multi-level dependencies," in *Sigcomm*, 2007.

[3] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl, "Detailed diagnosis in enterprise networks," in *SIGCOMM*, 2009.

[4] P. Rebreyend, F. E. Sandnes, and G. M. Megson, "Static multiprocessor task graph scheduling in the genetic paradigm: A comparison of genotype representations," tech. rep., 1998.

[5] R. Righter and S. H. Xu, "Scheduling jobs on non-identical ifr processors to minimize general cost function," in *Advances in Applied Probability, Vol. 23, No. 4, pp. 909-924*, 1991.

[6] S. Patil, M. Natu, V. Sadaphal, and H. Vin, "Analyzing batch processing systems using frequent subgraph discovery,"

[7] J. K. Lenstra and A. H. G. R. Kan, "Complexity of scheduling under precedence constraints," in *Oper. Res. vol. 26*, 1978.

[8] STG, "www.kasahara.elec.waseda.ac.jp/schedule/," 2011.

[9] DagGen, "www.loria.fr/ suter/dags.html," 2011.

[10] E. B. Fernandez and B. Bussell, "Bounds on the number of processors and time for multiprocessor optimal schedules," in *IEEE Transactions on Computers, Vol. C-22, NO. 8*, 1994.