Assigning Tasks in a 24-Hour Software Development Model

Pankaj Jalote, Gourav Jain
Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur, INDIA 208016
Email:*{jalote, gauravj}@iitk.ac.in* Fax:*+91-512-590725/590413*

Abstract

With the advent of globalization and the Internet, the concept of global software development is gaining ground. The global development model opens up the possibility of 24-hour software development by effectively utilizing the time zone differences. To harness the potential of the 24-hour software development model for reducing the overall development time, a key issue is the allocation of project tasks to the resources in the distributed team. In this paper, we examine this issue of task allocation in order to minimize the completion time of a project. We discuss a model for distributed team across time-zones and propose a task allocation algorithm for the same. We apply the approach on tasks of a few synthetic projects and two real projects and show that there is a potential to reduce the project duration as well as improve the resource utilization through 24-hour development.

Index Terms: Global software development, 24-hour software development, project scheduling, task allocation, dependency graph, optimal schedule.

1 Introduction

With the advent of globalization, many companies have started expanding their presence round the world, which has resulted in a single organization with multiple geographically distributed units. With the evolution of internet and other technologies, the communication among these distributed units has become easier and more efficient.

Organizations that have units in different locations are naturally considering models in which global software teams collaborate across distributed locations. This idea of global software development has evolved due to factors like cost advantage, availability of large labour pool, proximity of local market and conditions etc. Importantly, global software development also brings the possibility of a 24-hour development model where the different time-zones of different locations can be leveraged to reduce the total development time of a project, through daily handoffs of work [7].

The potential of 24-hour work day has been exploited to some extent in software maintenance. For example, there are many instances of products being supported in India where the customer in USA logs the complaint at the end of the day and finds the complaint resolved when the next morning - the resolution takes place in India when it is night in USA.

However, in development projects, global software development makes a project multi-site and multi-cultural and introduces a new set of communication, technical, managerial, and coordination challenges [2, 7]. Some experience seems to suggest that the communication and coordination difficulties may have a negative effect on the project schedule [5, 6, 11]. Some studies have been done to study the problem of communication in global software development, and many suggestions have been made to mitigate these[2, 5, 3]. Tool support has also been suggested to alleviate some of the problems [4].

Even if communication and coordination difficulties can be solved through suitable tools and technologies, can distributed software development provide benefits, and if so, to what extent? In this paper we address this question of what potential benefits can global software development bring in reducing the project execution time. For harnessing the benefits that may be possible, suitable communication and coordination systems will have to be put in place. An understanding of potential benefits can help evaluate whether the benefits are worth the costs of such systems.

We consider a 24-hour software factory model, where multiple teams that are geographically distributed across different time zones work on the *same project*. Intuitively, since this model supports a 24-hour operation, it should be possible to reduce the development time. However, the potential of improvement is heavily dependent upon the degree of inter-dependency between project tasks and nature of their various constraints. For example, if the tasks have fewer dependencies between them, then the development time depends primarily on the number of resources available, so a multi-site team provides no extra benefit

over a single-site team of the same size. Clearly, to maximize the reduction in completion time through the 24 hour model, task assignment needs to be done carefully, so that the project can be completed in minimum execution time while satisfying all the constraints.

In this paper, we first present our model for 24-hour development and then present a task scheduling algorithm. The algorithm takes the task model for a project and the set of available resources as input and produces a task-schedule with minimal schedule length. We then apply the approach to the task graphs of a few synthetically generated projects, and the task graphs of two real projects. We show that not only there is a substantial reduction in the completion time of a project, there is also an improvement of the resource utilization in the project.

2 Project Execution Model

We view a software project as a set of activities or tasks. A task is the smallest unit of work with a well defined functionality and external interface with other tasks. For example, a task could be developing a software module, writing a technical document, testing a piece of code or any other effort in the process of software development. A task is characterized by the total effort it needs (we assume that this is in person days). We also assume that the assignable tasks are such that they are assigned to one team member for execution, which is usually the case for the lowest level tasks in a project. We assume that the task effort includes all the time needed by the resource to perform the task, including the communication time, and that this effort is the same for the single-site development model, and the multi-site, 24-hour development model.

Software development is a process of orderly execution of these tasks. Since these tasks may have some relationship among them, they cannot be executed independently. The order of their execution is constrained by the inter-dependencies between them and their individual requirements. We consider the following possible constraints on the execution of a task:

1. **Operational Constraint:** A task τ_j is operationally dependent on another task τ_i if task τ_j can start only after the task τ_i completes its execution. For example, testing a module is operationally dependent on the coding of that module.

2. **Skill Constraint:** A task has a skill constraint for a particular skill, if it can be assigned only to those persons that possess the specified skill. For example, the task of database design has a skill constraint that it can be assigned to a person that has expertise in database. If a task does not have any skill constraint, then any resource can execute this task.

3. **Resource Constraint:** A task τ has a resource constraint for the task $\tau\prime$ if τ must be executed by the resource which has executed task $\tau\prime$. With this constraint, we cannot assign the resource to task τ until task $\tau\prime$ has been completed. This type of constraint usually comes in a project to maintain efficiency. For example, it is desirable to assign the task of enhancement of a software module to the person who had written that module.

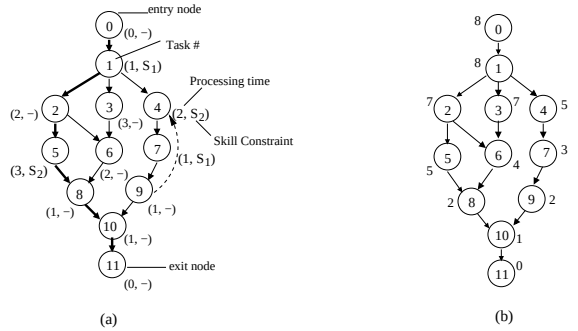

Figure 1: An example task graph

These constraints on the tasks in a project are the most common ones that project managers usually consider while scheduling. They need to be satisfied during the execution of tasks. We assume that all the constraints remain same in single-site and multi-site developments.

We represent the task set of a project by a task-graph which is a directed acyclic graph (DAG) $T = \{N, E\}$, where N is the set of nodes and E is the set of directed edges. Each node represents a task in the task set and edge represents the Operational dependency between tasks. For a node that represents a task τ_i, there is an association $(time(\tau_i), S_i)$, where $time(\tau_i)$ represents the time required for task τ_i for execution and S_i is the set of skills for which the task τ_i has the skill constraints. For simplicity, we assume that each S_i has atmost one skill. The null skill set is represented by the character '-'. The resource constraint of task τ_i for the resource that executed task τ_j is represented in the graph by a dotted arrow from task τ_i to task τ_j. We refer to the set of immediate predecessor tasks of a task τ_i as $P(\tau_i)$ and the set of immediate successor tasks of a task τ_i as $S(\tau_i)$.

Figure 1(a) shows a task graph having 12 tasks, each represented by a node. We assume that T has a single entry and single exit node. The entry node is considered as the predecessor of all nodes and the exit node as the successor

of all nodes. Both nodes are considered as dummy nodes with zero execution time.

The length of the path is the sum of execution times of all the nodes on the path. The weight of a node is the largest path length from the node to the exit node. In mathematical term, the weight of node τ is

$$weight(\tau) = \max_{k} \sum_{i \in \phi_k} time(i)$$

where ϕ_k represents the k^{th} path from the node τ to exit node and i represents a task on this path. The individual weight of each node of the example task graph is shown in Figure 1(b). The path from the entry to the exit node whose length is equal to the weight of the entry node is the *critical path* of the task graph. A critical path of the example task graph is shown by the dark arrows.

The weight associated with each task represents the minimum time the schedule must take from this task to complete the project. During scheduling, a task with a higher weight will get the priority if other conditions are satisfied. The selection of a node having the highest weight from the ready to execute nodes will ensure that the critical path of a project is being followed.

3 Resource Model

We assume that a 24 hour day is divided into three 8-hours time slots, with a different resource-set for each time slot. A resource-set comprises a set of individual resources $\{r_1, r_2 ... r_n\}$. Each resource (i.e. a person) has a set of skills associated with it which represents the skills that person has. The available resources can be modeled as a *resource-table*, in which each resource is identified by a number and for each resource, the resource-set it belongs to, the skills it has, and its working period, are specified.

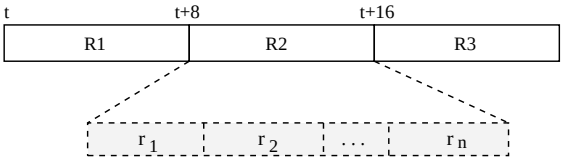

Figure 2: Resource Model

Figure 2 represents our resource model with the three resource-sets R1, R2, R3. Resource-set R2 only starts its work when R1 completes its time slot and similarly R3 only starts when R2 completes its time slot. For modeling simplicity, we are considering the time slots that do not overlap. In practice, however, an overlap between the time slots of two successive resource-sets is likely. This overlap can be used effectively for the handing over of tasks, providing clarification, and other activities that require interaction.

We also assume that all the resources needed by a project are assigned at the start of the project and are available throughout the project duration. Theoretically, the manpower buildup in a project follows a Rayleigh curve[12]. However, in practice, in a software project, the resources on a project undergo a few step increases or decreases[9]. And many times, particularly for smaller projects, resources are indeed all assigned in the start and freed at the end. The slack time available in such an allocation can be used for activities like training, documentation etc.

With these models for tasks and available resources, our objective is to assign tasks to the available resources such that the project completes in shortest duration. The benefit of the 24-hour development model will depend on how efficiently the schedule utilizes the advantage of different time zones of the resource-sets to reduce the project completion time.

4 Task Scheduling

Our task schedule algorithm is a heuristic based on the critical path method. It takes a task graph and a resource-table as input and generates a project schedule for the given project. The approach is to follow the critical path of the task graph for scheduling.

A ready queue Q of tasks is maintained which contains the tasks that are ready to execute. Initially, this queue contains all the immediate successors of the entry node. When a task completes its execution, all of its successors which are ready to execute are added to this ready queue.

Clearly, at a given time, only tasks in the ready queue can be assigned. However, instead of considering each task in the queue and assigning it, we propose a resource perspective and assign these resources to tasks in an attempt to keep the resources as busy as possible. The idea is that if the resource utilization is high then the task graph will be completed quickly.

The algorithm works as follows. Starting from time unit $t = 1$, for each time unit, it considers each of the three time slots in that time unit in order. In a time slot, it considers the available resources of the corresponding resource-set in order. A resource is available if it is not executing any task at the current time unit. For the first available resource r, it identifies two tasks from the ready queue$-$ τ_q, which is the highest weight task that has no skill or resource constraint, and τ_r, which is the highest weight task for which r

satisfies its skill or resource constraint. If both of the tasks τ_q and τ_r do not exist then the algorithm moves to the next available resource. In case if one of these tasks does not exist, then the other is allocated to resource r.

If both tasks τ_q and τ_r exist then one of them will be assigned to r. If τ_r has a higher weight, then it is assigned to r because not only does it represent the longest path but also because r satisfies its skill and resource constraints. If, however, τ_q has a higher weight, then assigning it to r might force task τ_r to wait for r till τ_q completes its execution. It might lead to increase in the schedule length. In an attempt to minimize the wait for τ_r, we compare $weight(\tau_q) - weight(\tau_r)$, which represents the difference between two path lengths, with the execution time of τ_q. If the execution time of τ_q is greater than the difference in weights then τ_r is assigned to r since otherwise waiting by the task τ_r will make the path length from τ_r to exit longer than the path from task τ_q and may increase the total completion time. On the other hand, if the difference of weights is greater than the execution time of τ_q, then task τ_q is assigned to r since the difference in path lengths will accommodate the waiting time of τ_r.

The algorithm is given in Figure 3. The final value of t gives the total time required to execute the whole project. This algorithm does not guarantee the optimal result but it should provide near-optimal solutions. This algorithm is based on the CP/MISF approach[10], which is regarded as the most effective heuristic approach for multiprocessor scheduling problem. Our task scheduling problem can be viewed as an extension of traditional minimal execution time multiprocessor scheduling problem [1]. With the combined approach of following the critical path and keeping the resources occupied, the final schedule length for the given project will reduce considerably.

Let us see the working of the algorithm on the example task graph of Fig 1(a). Since the entry and exit nodes are dummy nodes, we do not consider them for allocation. The weight of each task is shown in Figure 1(b). Suppose the available resources are as shown in resource-table of Table 1.

Resource #	Resource-Set	Skill-Sets	Work-period
l_1	R_1	S_1	00:00-08:00 GMT
l_2	R_1	S_2	00:00-08:00 GMT
m_1	R_2	-	08:00-16:00 GMT
m_2	R_2	S_2	08:00-16:00 GMT
n_1	R_3	S_1	16:00-00:00 GMT

Table 1: Resources for example task graph

At the start of first time slot of $t = 1$, we put task 1 in the ready queue and consider the resources l_1, l_2 of the

Inputs: Task graph T, Resource-Table.

Initialization: Put all the successors of the entry node in the ready queue Q. Set each resource as available.

Algorithm:
Repeat step 1 through step 6, for *t = 1,2,3 .... :*

1. Repeat step 2 through step 6, for each of the three time slots (with resource-sets R_1, R_2, R_3 respectively).
2. If any task τ completes its processing in previous time slot, then put all $\tau\prime \in$ S(τ) into ready queue Q provided P($\tau\prime$) has completed execution before this time-slot. Set the resource r which was executing the task τ, as available.
3. If $Q \neq \emptyset$ then repeat step 4 through 6 for all resources r of the resource-set of this time slot in order, provided r is available.
4. Let τ_q be the highest weight task in Q, which has no skill and resource constraint and τ_r be the highest priority task such that r satisfies its resource and skill constraint. (If two tasks have same weight then preference is given to task which has more immediate successors.)
5. If τ_q does not exist then assign τ_r to r. If τ_r does not exist then assign τ_q to r. Remove the assigned task from Q and set resource r as unavailable.
6. If tasks τ_q and τ_r both exist then,
a. if $(weight(\tau_q) - weight(\tau_r)) > time(\tau_q)$ then schedule τ_q to r; remove τ_q from Q and set r as unavailable.
b. else schedule τ_r to r. Remove τ_r from Q and set r as unavailable.

Figure 3: Task Scheduling Algorithm

resource-set R_1, in order. For the resource l_1, τ_r is the task 1 since this is the only task in ready queue and l_1 satisfies its skill constraint for skill S_1. As τ_q does not exist for l_1, thus task 1 is assigned to resource l_1 (step 5). We now consider the next available resource l_2. Since there are no ready tasks in ready queue, resource l_2 remains idle in this unit of time.

Now we move to the second time slot of $t = 1$. At the end of the first time slot, task 1 has completed its execution as it has one unit of execution time. So at the start of the second slot of time unit $t = 1$, task 2, 3 and 4 are in the ready queue. For the resource-set R_2, both resources m_1 and m_2 are available. For resource m_1, task 2 will be its τ_q since it has the highest weight as well as more number of immediate successor than task 3 (step 4). There is no τ_r for resource m_1, so task 2 is assigned to m_1. For resource m_2, τ_q is the task 3 and τ_r is the task 4. As *time(3)>weight(3)-weight(4)*, task 4 is assigned to m_2 (step 6b). At the start of third time slot of $t = 1$, task 3 is the only task in the ready queue. For resource n_1, task 3 is τ_q and there is no τ_r, so task 3 is assigned to n_1. This completes the first time unit. The schedule at the end of first time unit is shown in Figure 4(a). It shows the start and end of the execution of tasks in terms of time units.

At time unit $t = 2$, the ready queue will be empty at the start of first and second time slot since no task has completed its execution in respective previous time slots. At the end of second time slot, tasks 2 and 4 has completed their execution, so tasks 5 and 7 are ready for execution. As resource n_1 is not available, no task can be assigned in third time slot. Tasks 5 and 7 are assigned to resources l_2 and l_1 respectively at the first time slot in time unit $t = 3$, since they are τ_r for respective resources and there is no τ_q. At the second time slot of time unit $t = 3$, task 7 has completed execution in first time slot and hence its successor, task 9 is put in ready queue. At this time unit, both resources m_1 and m_2 are available. For resource m_1, there is no τ_q and τ_r in ready queue. For resource m_2, task 9 is τ_r since m_2 has executed task 4 and satisfied the resource constraint of task 9. Consequently, task 9 is assigned to m_2.

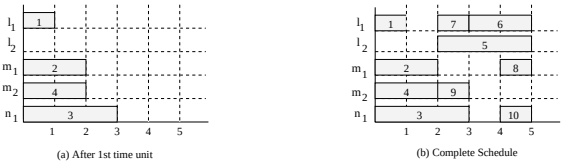

Figure 4: Task Schedule for example task graph

Similarly, tasks 6, 8 and 10 are scheduled and the final schedule is shown in Figure 4(b). The total schedule length is 5 time units. It is also the optimal solution for this problem.

% reduction →	$\leq 10\%$	$10\% - 20\%$	$> 20\%$
$p = 3$	62%	26%	12%
$p = 5$	38%	48%	14%
$p > 5$	27%	52%	21%

Table 2: Reduction in Schedule duration

5 Experimental Evaluation

In order to evaluate the effectiveness of the task scheduling algorithm, we did a few experiments. We generated about 100 task graphs randomly, with the number of tasks between 5 to 100. The time attribute of each task is uniformly distributed over the range of 1 to 20 time units. The number of tasks that have been assigned a skill or resource constraints varies from the 10 percent to 70 percent of the total number of task present in the task graph. The whole set of task graphs is tested for three different sizes of resource-sets, where the number of resources in each resource-set are 3, 5 and more than 5 resources respectively. We chose these resource-set sizes as we expect that a large project will be broken into relatively independent, smaller sub-projects at a top level, and the detailed scheduling that we are focusing on will be applied to these sub-projects, where the total team size is not likely to be too large. It should be pointed out that a resource set size of 5 implies that the total team size for that project is 15 in our model

We applied the algorithm on these task graphs and calculated the respective resultant schedule lengths. We also calculated the schedule lengths this algorithm generates if all the resources work at single location, and used that as the basis to determine the reduction in schedule the 24-hour model provides. The results are shown in Table 2.

The Table 2 shows, for different sizes of resource-sets, in what percentage of graphs the reduction in schedule is less than 10 percent, between 10 to 20 percent and more than 20 percent. As we can see, the benefit increases as the size of resource-set increases. The mean reduction in schedule length for the resource-set sizes 3, 5 and more than 5 are 11%,17% and 19% respectively.

We also studied the resource utilization in these projects. Resource utilization is the percentage of time the resources are occupied by project tasks. We found that the resource utilization also improved from 67% to 78%, from 61% to 68%, and from 54% to 63%, respectively, for the different resource-set sizes.

In order to study the potential benefits on actual projects, we took two actual software projects from Infosys Technologies Limited, a Bangalore based software organization. These projects have been executed in the traditional single-site manner and their detailed tasks and constraints are available. We took their actual task schedule, and scheduled it using our approach to determine the overall schedule. In other words, for comparison of schedules, we essentially simulated the execution of an actual single-site project.

The first project is the Weekly Activity Report (WAR)[8] system project and the second is ACIC development project[9]. The detailed project schedule of both projects were examined and various tasks and their dependencies were identified. The task graphs were then generated and various constraints related to each task were fixed. For effort, we used the estimated effort for each task as given in the project schedule. Activities like training were not included in the task graph. The projects were then scheduled in two scenarios when all the resources work in same time zone and when all the resources are evenly distributed (to the extent possible) in three resource-sets.

The duration of completion of different phases in the two scenarios is shown in Table 3 and Table 4 for the two projects. (An X means that the phase was not considered for detailed scheduling, mostly because these are phases where one or two people are involved and detailed scheduling is not an issue. They are there in planning largely because of effort estimation.) For the WAR project the overall reduction in schedule is from 88 days to 72 days. That is, the overall reduction in the completion time of the project by using the 24-hour model is 18%. Similarly, the overall reduction in the ACIC project is 21%. Note that the actual schedule of these two projects was 159 days and 140 days, respectively. We did not compare the schedule with the actual, as the actual schedule does not represent the best possible schedule for a single-site situation, but a possible schedule based on the actual resources available for the project. For a fair comparison, we have compared the schedule for the 24-hour model with the schedule that is generated for a single-site case if the same scheduling technique was used.

Phases	single resource-set	multiple resource-sets
Requirement Analysis	X	X
Project Mngt & Scheduling	X	X
Screen Prototyping	9	7
Functional Spec.	11	9
Sample Application	11	9
Architecture & Database Design	12	10
Detail Design	9	7
Building	16	12
Unit testing	8	6
System testing & Deployment	12	12
Overall	88	72

Table 3: Results for WAR Project

Phases	single resource-set	multiple resource-sets
Project Initiation	X	X
Scheduling and Mngt	X	X
Elaboration Iteration 1	22	15
Elaboration Iteration 2	10	7
Construction Iteration 1	8	6
Construction Iteration 2	7	6
Construction Iteration 3	7	5
System Testing & Deployment	18	18
Overall	72	57

Table 4: Results for ACIC Project

6 Conclusions

The 24-hour global development model is the approach where a distributed team works in different time-zones to establish a 24-hour workflow in a single project. In this paper we discuss the issue of scheduling of tasks to resources for reducing the overall execution time of a project.

We consider a 24-hour development model which consists of three teams working in 3 different 8-hour time slots. A software project is viewed as a set of activities or tasks. The project tasks have various constraints for its execution. Operational dependency of a task constraints its execution order with other tasks. Skill and resource constraint limits the resource space for a task for execution. The goal of task scheduling is to schedule these tasks for execution such that the constraints are satisfied and the graph is executed completely in the shortest possible time.

We presented a heuristic for scheduling based on the critical path method. It takes the task graph of a project and available resources as input and generates the minimal length project schedule for the project. We have tried the approach on some synthetic projects, and on two real-life projects. For these two projects, using their actual task sequence, we obtained the schedule for the single-site case and the schedule for the 24-hour model using our scheduling approach. For the examples, the approach provides a reduction of about 10% to 20%. However, the actual reduction in schedule depends on the nature of the graph.

There are many possible extensions of this work. Clearly the model can be enriched by taking the communication overhead explicitly into account, so as to model the extra communication cost of the distributed teams. We have worked with non-overlapping time slots. A more realistic model will consider time slots that are overlapping. In some projects there may be some tasks that have collocation requirement - i.e. engineers working on these tasks should be together [4]. The model needs to be extended to incorporate this requirement also. Other constraints that may exist in some situations, also need to be

modeled. Overall, we believe that much more work needs to be done to further enrich the model and then develop suitable scheduling algorithms for them. With a better understanding of the benefits and constraints, proper communication and coordination environments can be developed to support the tight coordination that is necessary to reap the benefits of the 24-hour model.

References

[1] I. Ahmad, Y.K. Kwok. "Static scheduling algorithms for allocating directed task graphs to multiprocessors". *ACM Computing Survey*. Vol. 31, No. 4, Dec,1999. pp. 406-471.

[2] E. Carmel. "Global Software teams: Collaborating across borders and time zones". *Prentice Hall, NJ*. 1999.

[3] E. Carmel and R. Agarwal, "Tactical approaches for alleviating distance in global software development", *IEEE Software*, March/April 2001, 22-29.

[4] C. Ebert and P. De Neve, “Surviving global software development” *IEEE Software*, March/April 2001, 62-69.

[5] J. D. Herbsleb and R.E. Grinter, "Splitting the organization and Integrating the Code: Conway’s Law Revisited" *Int. Conf. on Software Engineering* 1999, pp. 85-95.

[6] J. D. Herbsleb, et. al. "An emperical study of global software development: distance and speed", *Int. Conf. on Software Engineering*, 2001.

[7] J. D. Herbsleb and D. Moitra, "Global software development", *IEEE Software*, March/April 2001, 16-20.

[8] P. Jalote. *"CMM in Practice : Processes for Executing Software Projects at Infosys"*. Addison-Wesley, 2000.

[9] P. Jalote. *"Software Project Management in Practice"*. Addison-Wesley, 2002.

[10] H. Kasahara, S. Narita. "Practical multiprocessor scheduling algorithms for efficient parallel processing". *IEEE Transaction on Computers*. Vol. C-33, No. 11, Nov,1984. pp. 1023-1029.

[11] A. Mockus and D. M. Weiss, "Globalization by chunking: A quantitative approach", *IEEE Software*, March/April 2001, 30-37.

[12] L.H. Putman. "A general empirical solution to the macro software sizing and estimating problem." *IEEE Transactions on Software Engineering*, Vol 4, No. 4, 1978. pp.345-361.

[13] A. Repenning et. al., "Using components for rapid distributed software development", *IEEE Software*, March/April 2001, 38-45.