Optimized Execution of Business Processes on Crowdsourcing Platforms

Roman Khazankin, Benjamin Satzger, and Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
Argentinierstrasse 8/184-1, A-1040 Vienna, Austria
{lastname}@dsg.tuwien.ac.at
http://dsg.tuwien.ac.at/

Abstract—**Crowdsourcing in enterprises is a promising approach for organizing a flexible workforce. Recent developments show that the idea gains additional momentum. However, an obstacle for widespread adoption is the lack of an integrated way to execute business processes based on a crowdsourcing platform. The main difference compared to traditional approaches in business process execution is that tasks or activities cannot be directly assigned but are posted to the crowdsourcing platform, while people can choose deliberately which tasks to book and work on. In this paper we propose a framework for adaptive execution of business processes on top of a crowdsourcing platform. Based on historical data gathered by the platform we mine the booking behavior of people based on the nature and incentive of the crowdsourced tasks. Using the learned behavior model we derive an incentive management approach based on mathematical optimization that executes business processes in a cost-optimal way considering their deadlines. We evaluate our approach through simulations to prove the feasibility and effectiveness. The experiments verify our assumptions regarding the necessary ingredients of the approach and show the advantage of taking the booking behavior into account compared to the case when it is partially of fully neglected.**

Index Terms—**Human-centric BPM, Crowdsourcing, Incentive Management, Adaptive Process Execution**

I. INTRODUCTION

Today's fast changing business environments require companies to be highly flexible in order to stay competitive. Short, unpredictable business cycles and fluctuations, rapidly emerging technologies and trends, globalization, and the global interconnectedness provided by the Internet have increased the world's economical clock speed and competition among companies. Recent developments in IT promise to help companies to cope with the new requirements they are facing. *Social networks* can be leveraged to support people in loosely coupled and open team structures to efficiently collaborate. Web-based *crowdsourcing*, on the other hand, allows to broadcast tasks to a large network of people, the crowd, via an open call. Crowd members voluntary agree to solve crowdsourced problems motivated by incentives, such as money or social recognition. Crowdsourcing has the potential to give companies flexible access to a talent pool of almost unlimited size. In fact, according to an internal strategy document that leaked out in early 2012, IBM plans to employ a radically new business model [1]. It involves to let the company run by a small number of core workers. A dedicated Web-based platform is used to attract specialists and to create a virtual "cloud" of contributors. Similar to cloud computing where computing power is provided on demand, IBM's people cloud would allow to leverage a flexible on-demand workforce. Today's crowdsourcing systems are still relatively simple and only suitable for non-critical, atomic tasks requiring minor efforts. In particular, Amazon offers a task-based crowdsourcing marketplace called Amazon Mechanical Turk (AMT) [2]. Requesters are invited to issue human-intelligence tasks (HITs) requiring a certain qualification to AMT. The registered customers post mostly tasks with minor effort that, however, require human capabilities (e.g., transcription, classification, or categorization tasks [3]).

We foresee that in the future companies will increasingly use crowdsourcing to address a flexible workforce. However, it is still an open issue how to carry out business processes leveraging crowdsourcing. Most task-based crowdsourcing platforms simply provide requesters the possibility to publish simple task descriptions into a database all workers have access to. A task description in AMT for instance consists of a title, textual description, expiration date, time allotted, keywords, required qualifications, and monetary reward. Business processes or workflows, on the other hand, describe a logical structure between tasks crowdsourcing platforms cannot handle. The main problem is, however, that people book tasks voluntarily in crowdsourcing, which means the only way to influence booking and execution times of single tasks is to either change incentives or modify other aspects of a task, e.g., define a later deadline.

The contribution of this paper is an enterprise crowdsourcing approach that executes business processes on top of a crowdsourcing platform. For each single task in the business process we reason about the optimal values for incentive and time allotted when crowdsourcing them. The goal is to carry out the business process with minimal investments before the deadline. During execution of the business process we constantly monitor the progress and adjust these values for tasks that have not been booked by a worker yet. Our approach for calculating optimal values is based on mining historical data about task executions, e.g., which influence higher rewards have on the booking time, analyzing the current status of the business process, and quadratic programming, which is

a mathematical optimization formulation that can be solved efficiently. We evaluate our approach through simulations for different process sizes and structures. The experiments show the effectiveness of the approach and demonstrate its adaptivity to the poorly predictable crowdsourcing environment.

The paper is organized as follows. In Section II we introduce a motivating scenario for our work. Section III introduces the approach in detail. Results of conducted experiments are presented in Section IV. Related work is discussed in Section V. Finally, Section VI concludes the paper.

II. Motivating Scenario

We consider a scenario of a large software company that plans to install a enterprise-internal crowdsourcing platform for software development tasks. The platform allows employees to book tasks related to software development. Figure 1 schematically shows how tasks are presented to the employees, which is similar to most task-oriented crowdsourcing platforms like Amazon MTurk.

Type	Description	Ready to start	Effort	Time Allotted	Reward	
JavaScript	Click here	now	21h	5d	$2,090	Book
UI Design	Click here	Apr 1, 2pm	12h	7d	$810	Book
.NET	Click here	Apr 3, 9am	3h	14d	$110	Book
...	...	...	...	...	...	...

Fig. 1. Schematic UI for the enterprise-internal crowdsourcing platform of a large software company.

Each task is described by a type and a textual description. The third column gives an estimate when the employee will be able to start working on the task. Since a task may require input from other tasks the actual start date and time can deviate from the announced one. Effort provides information about the time effort necessary to finish the task. Time allotted defines the time frame in which the task is supposed to be processed. It starts when the task is ready to start or when the task is booked, whatever is later. The reward tells the employee how much he will get for successfully processing the task on time. Instead of money, rewards could also consist of more abstract reward points. When an employee books a task s/he is responsible for delivering the results within the allotted time. Tasks can be booked before they are ready to start. As long as tasks are not booked the system may modify allotted time and reward. The crowdsourcing platform generates and stores log information for each processed task, as illustrated in Figure 2. We assume that at least information regarding the task type, time effort, time allotted, the reward for which the employee actually booked the task, and the time it took from publishing to booking is stored for each task processed via the platform. Usually paying much for a task would reduce its booking time; also, having a high allotted time should make tasks more attractive compared to tasks with a tight deadline.

However, the software company has problems to map its business workflows to the crowdsourcing platform. Figure 3 shows a workflow describing a business process the company wants to execute. The aim of the process is to integrate a new

	Type	Effort	Time Allotted	Reward	Booking Time
	.Net	24h	14d	$520	1d
	.Net	12h	4d	$325	5h
	.Net	36h	5d	$570	6d
	.Net	36h	5d	$850	5h
	UI Design	12h	8d	$405	3h
	...	...	...	...	...

Fig. 2. The crowdsourcing platform maintains a database containing information related to the processing of each task.
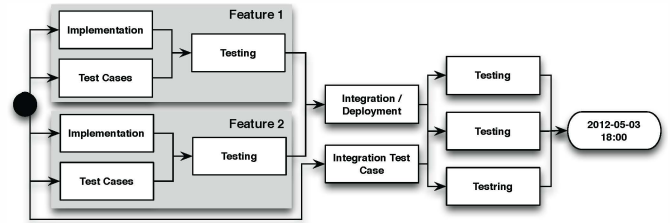

Fig. 3. An exemplary business workflow describing the development of a new plugin for a software product. The plugin consists of two features that need to be developed, integrated, and deployed into the software product. Each implementation step is followed by testing. There is a deadline for the completion of the whole plugin.

plugin into an existing software product. The plugin consists of two features that together make up the functionality of the plugin. Each feature consists of three tasks, the actual implementation and the writing of test cases, which can be done in parallel, and the testing of the implementation using the test cases. After both features have been implemented and tested, an integration and deployment step is necessary to ensure proper installation into the software product. Three different testing tasks are to ensure the high quality of the plugin; all three are based on the integration test case.

The introduced business process is simple yet helps to understand the challenges addressed by this paper. The question is how to crowdsource the tasks of the workflow using the crowdsourcing platform, i.e., how to set the values for the crowdsourced tasks. Type, description, and estimated effort of each single task typically are already available; the approximate ready-to-start times can be computed by the crowdsourcing platform once it has scheduled all predecessor tasks. This is relatively straightforward yet not trivial since it involves computation based on constant monitoring and recalculations to cover deviations from the schedule, e.g., delayed or early finished tasks. However, there is no obvious solution at all for how to determine the values for time allotted and reward. Time allotted should be assigned in a way to ensure the adherence to the deadline. Rewards should consider the usual "market prices" of the respective tasks, but also sometimes be increased to strengthen the competition among employees and ensure that tasks critical to the success of the workflow are timely booked. Also, in case the allotted time is short to process the task, then this should be reflected in the reward.

In this paper we introduce a novel algorithmic approach to crowdsource tasks that belong to a business workflow. We try to map tasks to the crowdsourcing platform such that the deadline is met and the aggregated rewards are minimized. In the next section we present our approach in detail.

III. APPROACH

The goal of our solution is to ensure the timely execution of business processes that contain crowdsourced tasks while minimizing the expenses associated with crowdsourcing rewards. We assume that the crowdsourcing platform allows to specify allotted time and reward for each task (as described in Sec. II). The expenses can be reduced by setting lower rewards for tasks, however, if the reward is too small, a task might stay not booked for too long, if booked at all. Such situations can significantly affect the execution of the process, and become a reason of missed deadlines. The allotted time also affects expenses: it is less likely that an employee decides to take an urgent task for a regular reward. Hence, there can be more employees interested in a non-urgent task at a lower reward, because some of them might have less experience in this type of tasks, and would like to improve, but need more time allotted. Obviously, the time allotted also directly influences the process completion time. The main idea of our approach is to find a most beneficial trade-off between rewards, allotted times, and expected booking times for crowdsourced tasks in the process. Specifically, we address the following questions:

How to estimate booking time? The time it takes someone to book a task after it is announced in the platform, or the *booking time*, can be influenced by various factors. However, we are convinced that it is driven mostly by the strength of the competition among employees. Therefore, tasks whose time allotted and reward combination satisfies demands of more people are generally booked earlier, and vice-versa. Undoubtedly, even if an employee is satisfied with the time allotted and the reward offered, s/he can still refuse to book a task, because of being too busy, not interested in this particular type of tasks, or just not being in the mood. Nevertheless, if the crowd is large enough, the trend should remain. Our approach is to determine this trend using platform logs (see Sec. II).

How to optimize allotted times and rewards? The optimization should consider the dependency between booking times, rewards, allotted times, and the structure of the process. Also, it can happen that something goes not as expected (e.g., a worker delays a critical task or completes it significantly earlier), so either it becomes necessary to get some tasks done faster to cope with a deadline or an opportunity to cut more costs emerges. The optimization therefore should perform adaptively, and consider the process state as well.

When should tasks be published in the crowdsourcing platform? If a task is booked by an employee, then the platform undertakes a commitment and can't demand the employee to perform faster or change the reward for this task any more. However, as mentioned above, an adaptive behavior can be advantageous, and it can be more beneficial to publish tasks later. But this should not be done too late and be aligned with the process execution state. Our approach is to publish a task when the sum of its optimized booking and allotted times, and expected execution time of subsequent activities in the process is almost as long as the time left before deadline. In Section IV we experimentally prove that such an approach produces optimal results.

We thus map the described functionality to components and propose a framework for a deadline-driven reward optimization for processes containing crowdsourced tasks (see Fig. 4). The estimator collects the statistical data from the platform logs and estimates the functional dependency for each type of task ①. The optimizer retrieves structure and state of processes ③, booking state of already published tasks ④, functional dependencies ②, and determines optimal values for booking time, reward, and time allotted. These values are further used by the publisher ⑤ which announces tasks to the platform at appropriate time and updates them if needed ⑥.
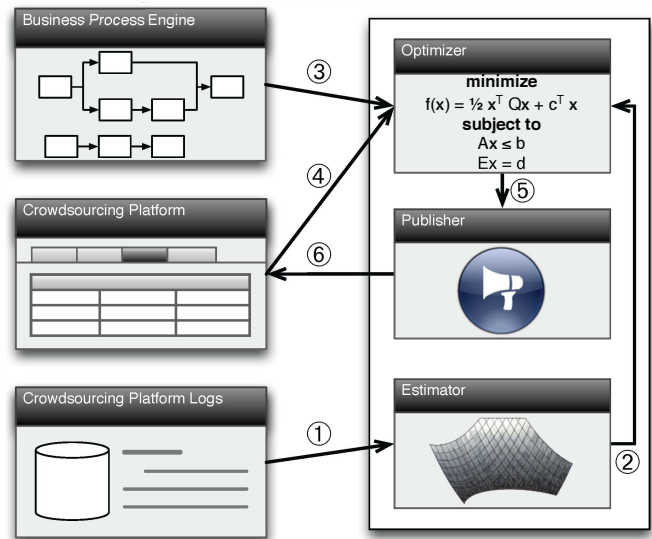

Fig. 4. The architecture of the framework

Subsections below provide a detailed description of the corresponding components of the framework. Estimation should be performed for all task types before rewards can be optimized. After some time, it can be re-executed to conform to the crowd's changing characteristics. The optimization and publisher are activated periodically thus realizing the adaptive behavior.

A. Estimator

As described in Sec. II, each log entry corresponds to a processed task and includes task type, weight (e.g., in hours of effort), allotted time, booking time, and reward. Let us reference to these values as T, w, t, bt, r. The estimation for each

task type is done independently. Therefore, for a particular type of task, an entry can be represented as $< w, t, bt, r >$.

Assuming that reward and time allotted linearly depend on the weight, and booking time depends on the combination of reward and time allotted, we can consider a mapping $bt' = f(t', r')$ populated with log entries as follows: $t' = t/w; r' = r/w; bt' = bt$, which reflects tasks with weight one.

Further, we need to estimate a function $ub(t, r)$ which is an upper bound for mapping f for each (t, r). Even with a weak competition, a task can be booked fast due to a coincidence. Therefore, for stable prediction in the context of deadline fulfillment, we are interested in the *maximum* time that it takes a task with specified reward and time allotted to get booked. The particular methods for upper bound estimation can vary according to the real setup.

Finally, using the discovered upper bound, we need to approximate the function $g(t, bt)$ that reflects the reward that needs to be set for a specified time allotted and expected booking time. This function will be used in an objective function for optimization. The dataset for approximating g is obtained as a set of tuples $< r_i, t_i, ub(t_i, r_i) >$ for each log entry. We argue that g should be approximated with a 2nd degree polynomial, because (i) polynomial-based optimization is well studied [4], (ii) many optimization frameworks support quadratic programming [5] , and (iii) the 2nd degree is a good trade-off between optimization complexity and fitting accuracy for the problem. In our experiments, the difference in accuracy between 2nd and 5th degree polynomial approximation was less than 5%, whereas it was more than 20% between 1st and 2nd degree. The estimation should also determine minimum and maximum values for all arguments. The approximated function should therefore have the following form:

$$g(t, bt) = a_1 * t^2 + a_2 * t * bt + a_3 * bt^2 + a_4 * bt + a_5$$

An example of an approximated function is illustrated in Sec. IV.

B. Optimizer

The goal of the optimization is to fulfill process deadline, while trying to minimize the offered rewards. Therefore, based on the state and structure of the process and estimated dependencies, we formulate a quadratic programming problem. The constraints ensure that the process can be finished before the deadline considering all the booking and allotted times, and the optimization objective is the sum of the rewards that will be paid for tasks contained in the process.

We formally represent a process as a directed acyclic graph, where nodes represent tasks and edges represent control flow. The graph has 2 special nodes that represent the beginning and the end of the process, namely *in* and *out*. Thus, for each node in the graph, there exists a path from *in* to *out* which contains this node. A task can be started when all its incoming edges are adjacent to already finished tasks. Besides crowdsourced tasks, a process can contain simple activities, whose execution times are regarded as constants. For simplicity, *in* and *out*

nodes can be considered dummy, i.e., simple activities with execution time 0. Such representation is more general than, e.g., combination of *flow* and *sequence* activities in BPEL. In this work we do not consider constructs such as conditions or loops for the sake of simplicity, although the approach can be extended to support such elements.

Each task has a property that indicates its status, which can be either `unavailable`, `published`, `ready`, `started` or `finished`. The process engine can thus launch only the tasks that are `ready`. Tasks are marked as `published` when published in the crowdsourcing platform, and change their status to `ready` when booked. For each not yet booked crowdsourced task, decision variables for allotted time and for booking time are included for optimization. The variables are restricted using the minimum and maximum values provided by the estimator. Two categories of constraints are included into the optimization model:

1) Constraints covering all the processing and allotted times throughout all possible process execution paths from the current state. The combination of these constraints ensures that the slowest branch will complete before the deadline.
2) Constraints covering booking time for `unavailable` and `published` tasks and all possible subsequent process execution paths. These constraints ensure that booking times will not endanger the deadline fulfillment.

If a task is already booked or represents a simple activity, then its execution time is fixed. If it has been already started, then its completion time can be estimated for current situation. In both of these cases, the execution time is regarded as a constant from the perspective of the optimization. The exact algorithm for building the constraints is described in Algorithm 1.

An example of the algorithm's functionality is shown in Fig. 5. A simple activity has already started and has been processed for five time units, as illustrated by the time line. Since the expected processing time for the activity is 20 time units, the optimizer assumes that the activity finishes in 15 time units. As there is only one started activity, the optimizer adds two constraints of first category because there exist two paths from this activity to *out* activity. As all the other tasks are either unavailable or published, the constraints of the second category should be created for them. Therefore, two constraints for booking time t_2 are generated, covering both successor paths. For both t_3 and t_4, only one constraint is generated for each single path to *out*.

The optimization objective is composed of a sum of rewards using the scaled values of estimated dependency functions:

$$min \sum_{s \in S} (g_{type_s}(t_s/w_s, bt_s) * w_s)$$

where S is the set of all tasks in the process, g_{type_s} represents the dependency function for the type of task s, t_s, bt_s are decision variables, and w_s is the weight of task s.

If the optimization problem turns out to be infeasible, the optimizer should try to extended the deadline and try again,

until a feasible solution is found. This will ensure that even if the deadline cannot be met, then the best possible solution will be provided.

Algorithm 1 Algorithm for creating optimization constraints

input : **timeToDeadline**, **processGraph**
call : **createConstraints** ([], **processGraph**.outNode)

createConstraints(list **path**, task **t**) {
  add **t** to the beginning of **path**;
  foreach (incoming edge **e** of **t**) {
    get adjacent node **t**' which is source of **e**;
    if (status of **t**' is not **finished**) **createConstraints**(**path**, **t** ');
  }
  if (there were no incoming edges with adjacent not **finished** nodes)
    addConstraint(**path**, 1st type); **else**
    addConstraint(**path**, 2nd type);
  remove **t** from **path**;
}

addTermsToExpression(list **path**, constraint expression **expr**) {
  foreach (**t** in **path**)
  if (status of **t** is **unavailable** or **published**)
    add decision variable for allotted time of **t** to **expr**; **else**
    add expected execution time left for **t** as a constant to **expr**;
}

addConstraint(list **path**, constraint type **cType**) {
  t = first task in **path**;
  if (**cType** is "2nd type" and
    status of **t** is not either **unavailable** or **published**) return ;
  create constraint expression **expr**;
  if (**t** is **unavailable**)
    add decision variable for booking time of **t** to **expr**;
  if (**t** is **published**)
    add predicted time for **t** to be booked as constant to **expr**;
  addTermsToExpression(**path**, **expr**);
  add optimization constraint [**expr** < **timeToDeadline**];
}
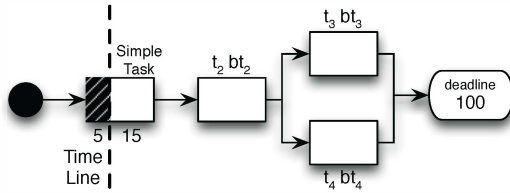
$$15 + t_2 + t_3 < 100$$
$$15 + t_2 + t_4 < 100$$

$$bt_2 + t_2 + t_3 < 100$$
$$bt_2 + t_2 + t_4 < 100$$
$$bt_3 + t_3 < 100$$
$$bt_4 + t_4 < 100$$

Fig. 5. Example for the generation of optimization constraints for the quadratic programming problem formulation.

C. Publisher

A task is published in the platform when the sum of its expected booking time, allotted time, and expected execution time of subsequent activities in the process is almost as current time to deadline. In other words, the time to deadline when it should be published can be determined by (i) finding the longest path from the task node to *out* node in the process graph, where weights of edges are set to the determined optimal values for time allotted of their source nodes, and (ii) adding the booking time of the task to this value. A task can also be updated after being published if it has not been booked yet.

In practice, the time to deadline value which is provided to optimizer and publisher can be lower than the real value in order to keep a small fraction of time reserved for handling unexpected events.

IV. EVALUATION

As we mention in Section V, best to our knowledge there are no similar approaches. Therefore, we were not able to make a comparative evaluation. Because the distinguishing feature of our approach is consideration of competition in crowdsourcing and booking time, we compare the effectiveness of the optimization with cases when booking time is partially or fully neglected. We also empirically prove the optimal choice of task publishing time, and evaluate the overall performance overhead of the optimization component.

To evaluate our approach, we examined a prototype implementation of the framework (accessible online[1]) in a simulated environment. We used MATLAB surface fitting for functional approximation, and GUROBI [6] framework for solving the quadratic optimization problem. We used a discrete time model, so time was measured in arbitrary integer units.

A. Simulation setup

Workers. The size of the simulated crowd was assumed to be 1000 workers. Platform logs were generated assuming that, at any point, only about 5% of them are willing to use the crowdsourcing platform (at various times those can be different workers). For every task type, each worker was assigned two values: the least time allotted that s/he needs to finish a task of this type with weight 1, and the minimum acceptable reward. These values were generated using the normal distribution. Then, for a random sampling of time allotted and reward pairs, 200 log entries were created. To pick a sensible booking time for a log entry, the competition value was calculated as a number of workers from the crowd, whose least time allotted and minimum acceptable reward were less than the log entry had. Then, assuming that only 5% of the potential competitors would actually compete for the task, the booking time estimation was guided by the probability that at least one of competing workers books the task before time n, assuming that the time of booking the task by one worker follows the normal distribution. The actual value was determined by stepwise increasing n and comparing a uniformly distributed random value with this

---

[1]http://www.infosys.tuwien.ac.at/prototype/incentivemgmt

TABLE I
TASK GENERATION PARAMETERS

Property	Type 1		Type 2		Type 3	
	Avg	Dev	Avg	Dev	Avg	Dev
Reward	100	15	50	7	80	10
Time allotted	20	3	15	3	13	2
Booking time for one worker	30	9	20	8.5	15	5

probability. Once it happens that the random value is less than the probability, n is the booking time. Such a method covered coincidental fast bookings while generally exhibiting the trend associated with competition. This method was also used to simulate actual booking while performing experiments, i.e., the crowd simulator was not informed about the booking time chosen.

Tasks. Three types of crowdsourced tasks were simulated. Each task was described by average reward and allotted time, booking time by one worker, and corresponding deviations. The types of tasks and their generation parameters are shown in Table I. The estimated dependencies between booking time, allotted time, and reward for tasks of Type 1 are illustrated in Fig. 6.
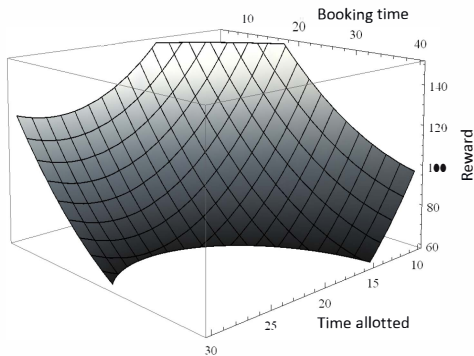

Fig. 6. Estimated quadratic polynomial that describes the dependencies between booking time, allotted time, and rewards for tasks of Type 1.

Processes. The simulator randomly generated processes with different sizes. We simulated small processes (5-10 tasks) and big processes (10-30 tasks), including all types of crowdsourced tasks and simple activities. We believe that bigger numbers are not realistic in a real setup, because usually business logic is clustered into concise compositions that are then managed on a higher level. Weights for tasks were selected randomly from the range [0.5,5].

B. Experiments

We ran the prototype in different simulation settings by randomizing process structures, and by emulating the inaccuracy of task execution and booking times (the results from different random generation seeds were averaged). The actual execution time for tasks was set using normal distribution with deviations of 0.1 and 0.2 of the supposed execution time. We

considered both cases when the deadline could and could not be adhered thus exploring how different parameters of the approach impact both critical and noncritical situations. We compare the results based on average reward, total time penalty (time penalty is a delay of a process with regard to the deadline for cases where deadline was missed), and number of missed deadlines produced, as these indicators fully reflect the goal of the approach.

Publishing time. In our solution a task is published when the difference between the time left before deadline and the sum of the task's allotted time, and expected execution time of subsequent activities in the process (let us refer to this value as *booking buffer*) is equal to its decided booking time. In order to prove that this is the optimal choice, we performed experiments where tasks were published at earlier and later times. The results are shown in Figure 7. We used booking buffer values equal to 0.2,0.5,1,2,3,4,5,6 multiplied by the task's decided booking time, and, finally, we tested the case when the tasks were published in the platform immediately after a process was started (*OnStart* mark in the figure).

It can be clearly seen that booking buffer equal to the expected (decided) booking time produces optimal results. When it is less than this value, the tasks are not booked on time, so the optimizer has to compensate that by putting higher awards, and, regardless of that, more deadlines are missed because of these delays. When booking buffer is greater than the decided booking time, then there is less room for maneuvering to handle uncertainty in execution and booking times, because tasks become booked earlier and their parameters cannot be changed anymore. This results into more missed deadlines and bigger penalties. This behavior then gradually changes in an opposite way, which can be explained by the fact that the real booking time can be longer than the estimated one, and therefore the impact of this inaccuracy is reduced for bigger values of booking buffer. However, the number of missed deadlines remains at least 25% greater than in the ultimate case when all the tasks are published when the process is started; rewards and time penalties in this case are almost the same.

Booking time. Consideration of booking time is a key feature in our approach. To analyze the effectiveness of this feature, we compare the full-featured optimization to the case where booking time constraints (second type of constraints, see Section III) are completely removed from the optimization problem (tasks are still published at the appropriate time according to the expected booking time which inferred from the decided rewards and allotted times), and to the case where average booking time is chosen for each task. The results for small and big processes are shown in Figures 9 and 10 respectively, depicting the same indicators as in the previous set of experiments.

As the results show, choosing an average booking time always results in approximately 13% more expenses for all process sizes, and produces more or almost as much penalties and deadline misses as the full-featured optimization does.
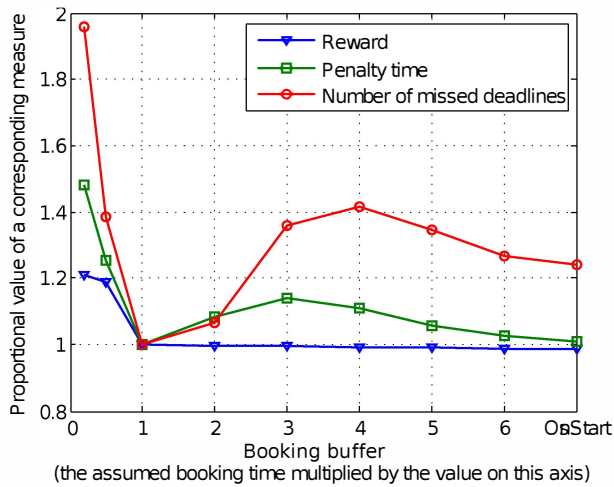
Fig. 7. Publishing time. This figure shows the effect when the booking buffer is varied and tasks are published not as suggested by our approach (1 on the x-axis).
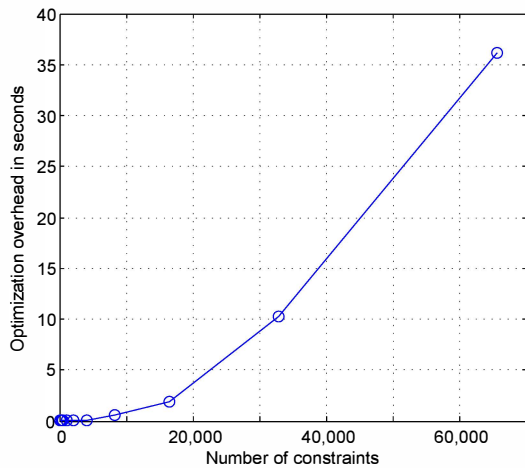

Fig. 8. Performance overhead. This figure shows the dependency of the performance overhead on the number of constraints in the optimization problem for one run.

This happens because some tasks do not need to be booked fast, and the full-featured optimization would pick less competitive and therefore less expensive values for rewards.

Disabled booking time constraints do not affect the indicators for big processes. This can be explained by the fact that there is almost always enough time for booking the tasks in long processes. Only first tasks of the process can be delayed, but it does not affect the overall performance. Also, booking times are always chosen to be maximal in this case because they are not constrained and it reduces the paid rewards. However, for smaller processes, the consideration of booking time is crucial, as it can stronger affect the relatively short process execution time. The unconstrained case produces 14% more deadline misses and penalties.

One can argue that the full-featured optimization should perform at least with the same performance as two other

approaches. In perfect conditions this assertion would hold. However, on the one hand, the booking time is estimated as an *upper bound*, therefore, the booking can often take less time than predicted. On the other hand, when the competition is too low, it can have the opposite effect: a task, which is assumed to be booked and executed earlier and costs more, can be eventually delayed more than a task which costs less and was expected to be booked later. Therefore, estimating realistic booking times is one of the key requirements of this approach. The accuracy of booking time in our experiments was 92%, which resulted in at most 2% of more missed deadlines and penalties.
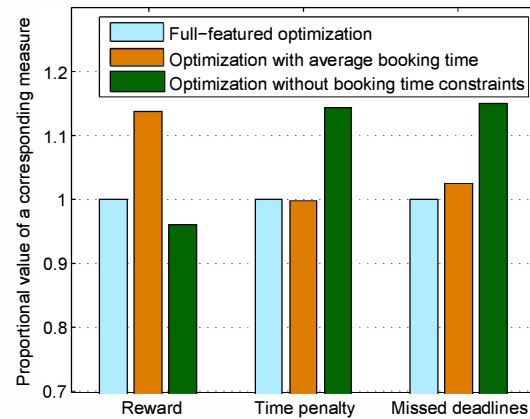

Fig. 9. Effect of neglecting the booking time in optimization for small processes (5-10 tasks).
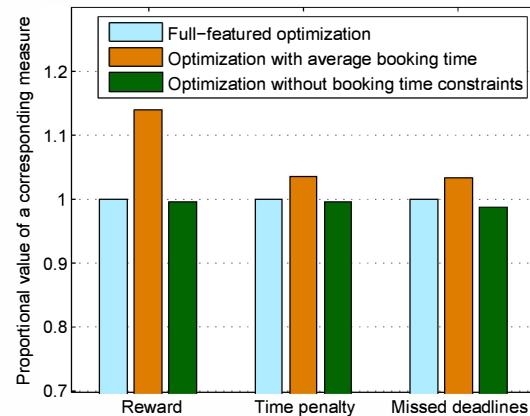

Fig. 10. Effect of neglecting the booking time in optimization for big processes (10-30 tasks).

Performance overhead. The overhead of optimization depends on the number of variables and the number of constraints. However, the number of variables for our problem

449

is proportional to the number of tasks involved, which was less than or equal to 30, and the variance within this limit did not affect the overhead. However, the number of constraints is proportional to the total amount of all possible paths that go through *in* and *out* activities (see Section III), and this number depends on the process structure and scales from one to tens of thousands. Figure 8 depicts the overhead dependency on the number of constraints for one optimization run[2].

For small processes (up to ten tasks), the worst case is when there is a sequence of five constructs each with two activities in parallel, the number constraints is less than $2^5 * 2 = 64$, which implies that an optimization run for a small process always takes less than 0.01 second. For bigger processes, the worst case is $2^{15} * 2 = 65535$, so an optimization run can take up to 35 seconds in this case. In both cases, the overhead is acceptable for performing periodical adaptations in processes with human tasks which can span from several minutes to hours or days.

C. Discussion

The results clearly show that booking time should be considered when publishing tasks to achieve the best adaptable behavior, because the best results are achieved when the booking buffer equals to the estimated expected booking time. Booking time constraints for optimization are not always favorable, however. They should not be used for bigger processes (more than ten tasks in out setting), but become important for smaller processes. Such smaller processes can emerge in, e.g., agile software development environments, where work is organized into short cycles with small sets of tasks.

V. RELATED WORK

In this work we combine crowdsourcing and business process execution. Major industry players have been working towards standardized protocols and languages for interfacing with people in a service-oriented way, which may be used as technical foundation for implementing our ideas in real businesses. Specifications such as WS-HumanTask [7] and BPEL4People [8] have been defined to address the lack of human interactions in service-oriented businesses [9]. Although some prospective features and possible extensions for dynamic resource management were outlined for these standards [10], [11], however, they have been designed to model interactions in closed enterprise environments where people have predefined, mostly static, roles and responsibilities.

The area of QoS-aware composition of Web services has many similarities to the topics addressed in this work. Web service compositions create value-added services by composing existing ones. Here the question arises which services to chose for participation in a composite service, given that there are many available Web services providing equivalent functionality. Liangzhao Zeng et al. [12] propose a QoS-aware middleware for selecting Web services that maximize user

satisfaction modeled as utility functions. They define multiple quality criteria, i.e., execution price, execution duration, reputation, successful execution rate, and availability. The authors propose service selection based on local optimization and global selection, considering aforementioned quality criteria. In the local optimization case service selection is done for each task individually, while the global planning also considers the interrelations between services. Integer Programming is used to solve the global planning problem. Canfora et al. [13] argue that genetic algorithms, while being slower than integer programming, represent an alternative, more scalable option. Another work [14] focuses on the optimization of large-scale QoS-aware compositions at runtime based on QoS specification based on constraint hierarchies. Multiple well-known metaheuristic optimization approaches are applied to solve the optimization problems. The major difference between Web service composition and crowdsourcing of business processes is that Web services which are to execute a functionality can be directly chosen while humans in the crowd are self-determined and act autonomously.

The recent trend towards *collective intelligence* and crowdsourcing can be observed by looking at the success of various Web-based platforms that have attracted a huge number of users. Well known representatives of crowdsourcing platforms include Yahoo! Answers [15] (YA) and the aforementioned AMT [2]. The difference between these platforms lies in how the labor of the crowd is used. YA, for instance, is mainly based on interactions between members. Questions are asked and answered by humans, thereby lacking the ability to automatically control the execution of tasks. In contrast, AMT offers access to the largest number of crowdsourcing workers. With their notion of HITs that can be created using a Web service-based interface they are closely related to our aim of mediating the capabilities of crowds to service-oriented business environments. According to one of the latest analysis of AMT [3], HIT topics include, first of all, transcription, classification, and categorizations tasks for documents and images. Furthermore, there is also tasks for collecting data, image tagging, and feedback or advice on different subjects. Related work in the area of crowdsourcing also includes experiments about behavioral aspects [16], impact of incentives on the performance of crowdsourcing [17], and impact of dynamic learning environments on the quality of crowdsourced tasks [18]. These studies partially confirm our assumptions (e.g., that higher incentives generate more interest and produce results faster), however, none of these works focus on internal enterprise crowdsourcing, and none of them discuss the controlled execution of business processes using such platforms.

While this paper focuses on how to take a workflow and optimally convert the subtasks into crowdsourcing tasks, there is also research about how to map crowdsourcing tasks to suitable workers. A possibility is to use auctioning mechanisms for implementing such a mapping [19], [20]. All workers

---

[2]Hardware used: Intel Core 2 Quad 2.40 GhZ with 6 GB of RAM

that meet the minimum requirements for a particular are invited to submit a bid to an auction created for assigning the task. The winning bid is determined by a combination of the workers' suitability for the task and the bids' prices. For improved reliability and training of workers a single task may be crowdsourced multiple times. Another possibility is to use scheduling [21] and specifically assign most suitable workers to pending tasks, while allowing workers to be flexible and to choose periods when they are available for work.

VI. Conclusions and Future Work

In this paper we presented an approach that allows to adaptively execute business processes on top of an internal competition-based crowdsourcing platform. The main feature that distinguishes our approach from other workflow and process optimization methods is consideration of time that it takes a crowd to book a task. We proposed a method for estimating the functional dependency of booking time by using statistical data, presented an algorithm for constructing an optimization problem, and empirically determined the optimal task publishing technique.

The results show that our model is effective for adapting the properties of tasks in a crowdsourcing platform to adhere to process deadlines and to minimize the rewards. We discovered that booking time should be considered when publishing tasks to achieve the best adaptable behavior, and that taking booking time into account in optimization can reduce deadline misses up to 14%. The approach can also be used to predict feasibility and expenses for a specified deadline by running a simulation like ours, therefore allowing to explicitly observe the tradeoff between processing time and associated costs.

It is fair to notice that the approach assumes booking time to be comparable to execution time of tasks. So, for example, if there is a strong overall competition in a crowdsourcing platform and tasks are taken not long after being published, then booking time might have a weaker effect on the process. It can be also discovered that booking time depends on the weight of a task. Such a dependency can be mined from logs and can be handled by our model as well.

In a real scenario, the results will heavily depend on the accuracy of booking time dependency approximation. Also, we do not consider "unbooking" or the possibility of re-negotiating reward/allotted time with a worker when the task is already booked. Such extensions might be an interesting direction for future work. Also, it would be interesting to compare our approach to auction-based crowdsourcing (e.g., [19]) and to extend it to support custom penalty functions.

Acknowledgement

References

[1] Spiegel Online (In German), http://www.spiegel.de/wirtschaft/unternehmen/0,1518,813388,00.html, last access Sep. 2012.
[2] Amazon Mechnical Turk, http://www.mturk.com, last access Sep. 2012.
[3] P. G. Ipeirotis, "Analyzing the Amazon Mechanical Turk Marketplace," *SSRN eLibrary*, vol. 17, no. 2, pp. 16–21, 2010.
[4] Z. LI, "Polynomial optimization problems, approximation algorithms and applications," Ph.D. dissertation, The Chinese University of Hong Kong, 2011.
[5] J. Nocedal and S. Wright, *Numerical Optimization*, 2nd ed., ser. Springer Series in Operations Research and Financial Engineering. Springer, 2006.
[6] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2012. [Online]. Available: http://www.gurobi.com
[7] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau *et al.*, "Web services human task (ws-humantask), version 1.0," *available at http://incubator.apache.org/hise/WS-HumanTask_v1.pdf*, 2007.
[8] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic, "Ws-bpel extension for people–bpel4people," *Joint white paper, IBM and SAP*, 2005.
[9] F. Leymann, "Workflow-based coordination and cooperation in a service world," in *Cooperative Information Systems (CoopIS '06)*, ser. LNCS. Springer, 2006, pp. 2–16.
[10] N. Russell and W. M. Aalst, "Work distribution and resource management in bpel4people: Capabilities and opportunities," in *International conference on Advanced Information Systems Engineering (CAiSE '08)*. Berlin, Heidelberg: Springer, 2008, pp. 94–108.
[11] D. Schall, B. Satzger, and H. Psaier, "Crowdsourcing tasks to social networks in BPEL4People," *World Wide Web, Springer*, 2012, 10.1007/s11280-012-0180-6.
[12] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *Software Engineering, IEEE Transactions on*, vol. 30, no. 5, pp. 311 – 327, may 2004.
[13] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for qos-aware service composition based on genetic algorithms," in *Genetic and evolutionary computation (GECCO '05)*. New York, NY, USA: ACM, 2005, pp. 1069–1075.
[14] F. Rosenberg, M. B. Müller, P. Leitner, A. Michlmayr, A. Bouguettaya, and S. Dustdar, "Metaheuristic Optimization of Large-Scale QoS-aware Service Compositions," in *International Conference on Services Computing (SCC'10)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 97–104.
[15] Yahoo! Answers, http://answers.yahoo.com/, last access Sep. 2012.
[16] G. Paolacci, J. Chandler, and P. Ipeirotis, "Running experiments on amazon mechanical turk," *Judgment and Decision Making*, vol. 5, no. 5, pp. 411–419, 2010.
[17] W. Mason and D. J. Watts, "Financial incentives and the "performance of crowds"," in *SIGKDD Workshop on Human Computation (HCOMP '09)*. New York, NY, USA: ACM, 2009, pp. 77–85.
[18] J. Le, A. Edmonds, V. Hester, and L. Biewald, "Ensuring quality in crowdsourced search relevance evaluation: The effects of training question distribution," in *SIGIR Workshop on Crowdsourcing for Search Evaluation*, 2010, pp. 21–26.
[19] B. Satzger, H. Psaier, D. Schall, and S. Dustdar, "Stimulating Skill Evolution in Market-Based Crowdsourcing," in *9th International Conference on Business Process Management (BPM '11)*, ser. LNCS. Springer, 2011, pp. 66–82.
[20] ——, "Auction-based crowdsourcing supporting skill management," *Information Systems*, 2012, 10.1016/j.is.2012.09.003.
[21] R. Khazankin, H. Psaier, D. Schall, and S. Dustdar, "Qos-based task scheduling in crowdsourcing environments," in *International Conference on Service-Oriented Computing (ICSOC '11)*, ser. LNCS. Springer, 2011, vol. 7084, pp. 297–311.