

Algorithms for Automating Task Delegation in Project Management

Bogdan Pop

Department of Computer Science
Babes-Bolyai University

Kogalniceanu 1, 400084, Cluj-Napoca, Romania
Email: popb@cs.ubbcluj.ro, bogdan.pop@webprator.eu

Florian Boian

Department of Computer Science
Babes-Bolyai University

Kogalniceanu 1, 400084, Cluj-Napoca, Romania
Email: florin@cs.ubbcluj.ro

Abstract—Project management can be defined as a complex set of activities that are performed by project managers, individuals or larger entities, that requires proper application of skills, knowledge, tools and techniques in order to reach or exceed project requirements. Two of the most important skills or techniques that greatly influence the end result of a project are task delegation and resource allocation. Poor decisions while delegating tasks or planning resources' allocation can result in defective project results. Many project management applications and models aid project managers in proper task delegation and resource allocation. This paper presents models and task delegation algorithms that can automate task assignment, thus reducing manual delegation, reducing loss and improving projects' end results.

I. INTRODUCTION

DURING a project's lifecycle many factors can change at any given point in time. In order to preserve the scope and objectives of the project and its sub-projects, developers have to take countermeasures swiftly. A single change during a task or an unforeseen event can trigger a chain reaction and derail greatly the project's development. Changing the terms and environment of the project can also add additional risks and the development team must be able to assess the situation quickly and make the proper adjustments in order to deliver the project successfully. The longer it takes for such countermeasures to be considered and performed the losses are likely to be higher and growing.

Therefore projects should be proactively managed by continuously improving and detailing the plan of action as more detailed and specific information and accurate estimations become available during the project's lifecycle. To achieve this, the project managers and project owners and stakeholders need to easily and fully grasp all aspects of the project. The bigger the project the harder it is to generate reports and project statuses. This is why it is recommended that developers use proper project management applications for their projects, allowing more focus to actual work than planning and calculating reports and project statuses. Furthermore, project management applications simplify a variety of tasks that project managers must perform with the help of dedicated tools and features [1][3].

Since task delegation and resource allocation are one of the most important aspects that greatly influence the outcome of a

project, it is clear that simplifying, streamlining and possibly automating task delegations and resource allocations within projects would have positive impacts on overall results, costs and profits.

A concept application that automatically assigns, without any human intervention, newly created tasks within a project and its deployment model have been previously presented in [4]. The end goal of the concept application was to ease the project manager's job, to minimise costs and maximise resource usage to its fullest. The paper presented the database model, the base algorithm and a deployment scenario as software as a service of the presented concept application. The algorithm was used to reduce the role of a project manager as known today, allowing the usage of the project manager's knowledge for actual development.

This paper is structured as follows. Section 2 briefly describes the current proposed web application, its distribution model and task delegation mechanism presented in [4] which have shown promising results while being tested and studied in comparison to some other popular project management applications currently available on the market [5]. Section 3 presents enhancements and additions that can be applied to the noSQL model and to the task delegation algorithm that may improve the performance of the system and its yielded results. The 4th section describes additional tests and studies that can be performed on the amended system to assess if the changes made have improved the performance of the system or not. The final two sections present future developments and conclusions, respectively.

II. CURRENT STATE OF THE TASK DELEGATION MODEL AND ASSIGNMENT ALGORITHM

Achieving automation in the task delegation process can be done by using a number of methods, most of them quite new and derived from the field of artificial intelligence. This includes, but is not limited to neural networks, evolutionary algorithms or swarm intelligence algorithms.

However, the application developed based on the proposed model [4] can potentially store vast amount of information, which would complicate the A.I. algorithms, especially with respect to computation times. Since time management is a critical part of project management and plays an important

```

1 {
2   "task name": {
3     "keywords":
4     [
5       { "keyword_name_1": "optional_priority" },
6       { "keyword_name_2": "optional_priority" }
7     ]
8   },
9   "task details":[
10    { "username": "username value" },
11    { "project": "project name" },
12    { "timeToComplete": "time" },
13    { "completed": "date time" },
14    { "deadline": "date time" },
15    { "finishedOn": "date time" },
16    { "assignedOn": "date time" }
17  ]
18 }

```

Fig. 1. JSON representation of the Tasks super column as presented in [4]

role in a project's success, A.I algorithms may not be well suited for the task.

Different approaches such as the Gale-Shapely Courtship Algorithm described in [2] have been taken into consideration as well. However, the chosen model was the usage of classic iterative algorithms that create automation with respect to task delegation by applying a set of instructions to properly stored, sorted and indexed data available on the project's backend database.

Project data is being distributed across multiple nodes via NoSQL databases, specifically Apache Cassandra [6]. A case of why NoSQL is better suited than classic relational databases is also presented in [4].

The initial proposed model used NoSQL databases to store information regarding project tasks, their types, the people that have worked on them, the time required to complete each task, personnel availability for future tasks and more. An algorithm that used this data to programmatically determine the best match for a newly added task was created. Figures 1, 2, 3 and 4 present the original schema of the most important column families in the database while the initial task delegation method is presented by Alg. 1.

Fig. 1 stores information regarding the undergoing tasks of the project, such as metadata keywords, comprising project, deadline, if it were already assigned or finished or not. Fig. 2 stores the availability times of each user. Since the chosen database system has a default lexicographic indexing, each user's availability is stored by using a date and username key with its two parts separated by a hash tag. The value stored is the time during a day when a particular user is available. The performance score and number of tasks of each user based on task metadata keywords is also stored as shown in Fig. 3.

By automating the task assignment process the project manager was no longer required to manually perform delegations and was able to have a more direct impact in the actual development of the project, instead of only leading it. Moreover, programmatically assigning tasks resulted in fewer errors compared to those made by a human project manager. Therefore, the development costs and time required to complete projects were reduced [4].

Algorithm 1 Original task delegation algorithm as presented in [4]

```

def function addNewTask(task)

    foundUsers = null
    iter = 1
    taskAssigned = false

    while ( taskAssigned==false && iter <10 )
    do

        resetOverallScore(foundUsers)

        for keyword in task.keywords do

            foundUsers.push(
                getUsersWithBestScore_taskAssign(
                    keyword,
                    Start = 10*iter -9,
                    End = 10 * iter)
            )

            for user in foundUsers
                userScore =
                    GetKeywordScore_userScores(
                        keyword,
                        user)
                user.overallScore += userScore
            end
        end

        foundUsers.sort_by { overallScore }

        count = 0
        while ( taskAssigned==false &&
            count < foundUser.size )
        do
            if foundUser[count].isAvailable?
                addTaskToUser(
                    foundUser[count],
                    task)
                taskAssigned=true
                return true
            end
        end

        iter = iter + 1
    end

    return false
end

```

```

1 {
2   "DATE_1#username_1":[
3     { "startTime_1":"endTime_1" },
4     { "startTime_2":"endTime_2" }
5   ],
6   "DATE_1#username_2":[
7     { "startTime_1":"endTime_1" },
8     { "startTime_2":"endTime_2" }
9   ],
10  "DATE_2#username_1":[
11    { "startTime_1":"endTime_1" },
12    { "startTime_2":"endTime_2" }
13  ]
14 }

```

Fig. 2. JSON representation of the userAvailability column family as presented in [4]

```

1 {
2   { "keyword_1#username_1": "score_1#tasksNo" },
3   { "keyword_1#username_2": "score_2#tasksNo" },
4   { "keyword_1#username_3": "score_3#tasksNo" },
5   { "keyword_2#username_4": "score_1#tasksNo" },
6   { "keyword_2#username_1": "score_2#tasksNo" },
7   { "keyword_2#username_2": "score_3#tasksNo" },
8   { "keyword_2#username_5": "score_4#tasksNo" },
9   { "keyword_3#username_1": "score_1#tasksNo" }
10 }

```

Fig. 3. JSON representation of the userScores column family as presented in [4]

III. ENHANCEMENTS TO THE CURRENT MODEL AND ALGORITHM

The initial algorithm described in [4] and presented in Alg. 1 worked in the following manner: when a new task was created the algorithm looped through the taskAssign column family (Fig. 4) for each keyword, starting from top to bottom, from the best score to the lowest possible, in order to assign it to the best suited user. The algorithm then computed each user's overall score counting zero if a user's score was null for a specific keyword. Following that, the user that got the best score and was available for work, according to its userAvailability (Fig. 2) column data, was assigned the task. If the user with the best score could not complete the task on time, the next user with the best score lower than the previous user's score was selected. Finally, if no suited users were found, the task remained unassigned and the project manager had to manually perform the delegation.

The study performed on the model and presented in [5] revealed the aforementioned un-assignment issue. On the first projects and for the first tasks within them, no users were selected since not enough data on their performance was stored in the database. The proposed modifications are designed to solve this issue such that all tasks, no matter their creation time, initial phases of the project, during the project or near its closing, are all automatically assigned by the proposed concept

```

1 {
2   { "keyword_1#score_1#tasksNo1": "username_1" },
3   { "keyword_1#score_2#tasksNo1": "username_2" },
4   { "keyword_1#score_3#tasksNo1": "username_3" },
5   { "keyword_2#score_1#tasksNo2": "username_4" },
6   { "keyword_2#score_2#tasksNo2": "username_1" },
7   { "keyword_2#score_3#tasksNo2": "username_2" },
8   { "keyword_2#score_4#tasksNo2": "username_5" },
9   { "keyword_3#score_1#tasksNo2": "username_1" }
10 }

```

Fig. 4. JSON representation of the taskAssign column family as presented in [4]

```

1 {
2   { "skillset_1#keyword_1#username_1": "score_1" },
3   { "skillset_1#keyword_1#username_2": "score_1" },
4   { "skillset_1#keyword_1#username_3": "score_1" },
5   { "skillset_1#keyword_2#username_2": "score_2" },
6   { "skillset_2#keyword_1#username_1": "score_1" },
7   { "skillset_2#keyword_1#username_3": "score_1" },
8   { "skillset_2#keyword_2#username_1": "score_2" },
9   { "skillset_2#keyword_2#username_3": "score_2" }
10 }

```

Fig. 5. JSON representation of the userSkillsetScore column family

```

1 {
2   { "keyword_1#username_1": "preference_score" },
3   { "keyword_1#username_2": "preference_score" },
4   { "keyword_1#username_3": "preference_score" },
5   { "keyword_2#username_1": "preference_score" },
6   { "keyword_2#username_2": "preference_score" },
7   { "keyword_2#username_3": "preference_score" },
8   { "keyword_3#username_1": "preference_score" },
9   { "keyword_3#username_3": "preference_score" }
10 }

```

Fig. 6. JSON representation of the userPreferenceScore column family

application and no manual input from the project manager is required.

In order to achieve the proposed scope, the cause of the problem had to be determined. The issue was in fact lack of data within the database, so the solution was to pre-populate the database with relevant information regarding each user. This could theoretically be possible for different projects within the same company or entity. However, this may not be the case for each developed project and it is therefore not a viable and feasible solution.

The proposed solution is a fallback within the task assignment algorithm that would be triggered when the original algorithm could not automatically assign a task based on data it can access from the userScores, taskAssign and userAvailability columns. This trigger would use two new column families that would store information about the users, mainly their skill-set, including all their certified and non-certified ones as well as their preference to what kind of work and tasks they prefer. Their skill-set information could be automatically computed by using predefined scores for different types of certifications. Their non-certified skills could only be scored and measured by an authority within the company or within the project, such as a HR or management representative. Fig. 5 shows the database schema for the user's skill-set score.

The mechanisms for generating a user's preferences score are trivial, each user having access to the project management application in order to set their preferences. The database schema of the preferences score is similar to that of the user's skill-set score and is presented in Fig. 6, while Alg. 2 presents the modified task assignment delegation process.

Another approach for modifying the algorithm would be to always take into account the skillset score and preference score of every user on the project. If this approach were chosen, a balance between users' past performance score, their skill score and preference score should be selected.

There are a couple of options for balancing the three different scores as follows. The first one is to allow manual

Algorithm 2 Modified task delegation algorithm with skill-set and preference fallback

```

def function addNewTask(task)
  foundUsers = null
  iter = 1
  taskAssigned = false
  while ( taskAssigned == false && iter < 10)
  do
    resetOverallScore(foundUsers)
    for keyword in task.keywords do
      foundUsers.push( getUsersWithBestTaskAssignScore(keyword,
        Start = 10*iter -9, End = 10 * iter) )
      for user in foundUsers
        userScore = GetScoreForKeywordFrom_userScores(keyword, user)
        user.overallScore += userScore
      end
    end
    foundUsers.sort_by { overallScore }
    count = 0
    while ( taskAssigned == false and count < foundUser.size ) do
      if foundUser[count].isAvailable?
        addTaskToUser(foundUser[count], task)
        taskAssigned=true
        return true
      end
    end
    iter = iter + 1
  end
  # trigger that takes skillset and preference into account
  while ( taskAssigned == false && iter < 10) do
    resetOverallScore(foundUsers)
    for keyword in task.keywords do
      foundUsers.push( getUsersWithBestSkillSetScore(keyword,
        Start = 10*iter -9, End = 10 * iter) )
      for user in foundUsers
        uSkillsetScore = GetScore_userSkillsetScore(task.skillset, keyword, user)
        uPreferenceScore = GetScore_userPreferenceScore(task.skillset, keyword, user)
        uSkillsetPreferenceScore = uSkillsetScore*0.75 + uPreferenceScore*0.25
        user.overallScore += uSkillsetPreferenceScore
      end
    end
    foundUsers.sort_by { overallScore }
    count = 0
    while ( taskAssigned == false and count < foundUser.size ) do
      if foundUser[count].isAvailable?
        addTaskToUser(foundUser[count], task)
        taskAssigned=true
        return true
      end
    end
    iter = iter + 1
  end
  return false
end

```

selections within each company and within each project. This way, management personnel could modify the ratio based on their preference and desired outcome within a project. The second option would be hardcoding the ratio for the three scores directly into the algorithm.

In order to obtain the perfect balance and ideal ratio, multiple tests with different ratios should be performed on the same set of tasks within identical projects in order to obtain relevant, comparable data. Alg. 3 shows the modifications required to use the balancing factor between users' performance, skill-set and preference scores, with their ratios being 50%, 25% and 25% respectively. The ratios were empirically determined as the algorithm allows modification for each project within different companies. Future work may include a more formal, mathematical approach to establishing the best ratios for the algorithm.

IV. TESTING THE PROPOSED MODEL AND ALGORITHM ENHANCEMENTS

The first step towards testing the modifications performed on the model and on the algorithm is to implement and deploy the changes into the web application developed based on the initial model [7]. The original model and its corresponding web application have already been tested and studied by using a small scale project developed by a small three-man development team [5].

The study revealed that the proposed concept is successful and overall performance improves with each new project that is managed via the application. The original study has been performed on a small scale construction project, a treehouse building process, and the results and outcomes may be completely different on larger projects and with larger teams or integrated ones. Therefore, future studies and tests of the enhanced model and task delegation algorithm should be performed by using larger, more complex projects.

The study presented in [5] also shown some drawbacks that interfered with the project workflow in the initial steps of the development, mainly because there was no information stored that could be used to automatically assign tasks to workers. The result of this test has generated the proposed enhancements presented in this paper. It is straightforward that if more tests are performed the likelihood to discover even more extensions and improvements that can be applied to the proposed concept will increase. It is therefore recommended that more diverse tests should be performed.

Additionally, tests should be performed on identical projects in order to obtain the golden ratio between the three scores: user performance, skill and preference, described in section 3 of the current paper, ratio that could be used for hardcoding the balancing factor within the task assignment algorithm.

V. FUTURE WORK

Proactive project management usually requires the following tasks be performed: identifying the requirements and objectives of the project, addressing the concerns and expectations of the project owners / stakeholders, balancing the project

Algorithm 3 Modified task delegation algorithm with performance, skill-set and preference balancing

```

def function addNewTask(task)
    foundUsers = null
    iter = 1
    taskAssigned = false
    while (taskAssigned==false && iter <10)
    do
        resetOverallScore(foundUsers)
        for keyword in task.keywords do
            foundUsers.push(
                getNUsersWithBestTaskAssignScore(
                    keyword,
                    Start = 10*iter -9,
                    End = 10 * iter)
            )
            foundUsers.push(
                getNUsersWithBestSkillSetScore(
                    keyword,
                    Start = 10*iter -9,
                    End = 10 * iter)
            )
        for user in foundUsers
            uPerformance =
                GetScore_userScores(keyword, user)
            uSkillset =
                GetScore_userSkillsetScore(
                    task.skillset, keyword, user)
            uPreference =
                GetScore_userPreferenceScore(
                    task.skillset, keyword, user)
            userScore = uPerformance * 0.5 +
                uSkillset*0.25 + uPreference*0.25
            user.overallScore += userScore
        end
    end
    foundUsers.sort_by { overallScore }
    count = 0
    while ( taskAssigned==false &&
        count < foundUser.size )
    do
        if foundUser[count].isAvailable?
            addTaskToUser(foundUser[count], task)
            taskAssigned=true
            return true
        end
    end
    iter = iter + 1
end
return false
end

```

constraints: scope, budget, schedule, resources, quality and risks [1]. The current proposed models and algorithms address only scheduling of personnel. An important aspect that can be improved upon is scheduling of hardware and other non-human resources.

Future developments may include enhancements of the current model such that tasks could be assigned to multiple individuals or such modifications that each task may or may not have preceding tasks to create proper dependencies. The current additions to the model allows tasks assignment based on user skill and preference. Similarly, a simple extension could be added such that more difficult tasks are assigned to individuals or teams with more experience and easier tasks assigned to the rest of the development team. This can be achieved if the model is modified in such a way that the database stores the difficulty of each task which could be assessed automatically or by the project manager. Similarly, one can also define some tasks as urgent, and these tasks should be assigned quicker based on their emergency level.

Future work may also include additional features such as estimating budget costs and automatically determining any risks before they affect the project. Another important aspect that would dramatically increase the potential of the application would be automatically identifying the requirements and objectives of the project from client communications, and translating them into tasks that developers understand and know how to perform.

VI. CONCLUSIONS

The proposed application concept reduces the role of the project manager by automatically delegating tasks and shows potential for large growth. Further more, the original application has performed up to 26.77% better than those of the competitors, as shown in [5]. The same test also shown that there is at least a 20% margin for improvement, leaving room for continuous and future developments.

The few cases when manual input by project managers was required [4] have also been reduced by the modifications performed on the original model and presented in this paper.

REFERENCES

- [1] *A guide to the project management body of knowledge (PMBOK Guide)*, Fourth Edition, Project Management Institute, Inc., 2008
- [2] D. Gale, *The two-sided matching problem. Origin, development and current issues*, International Game Theory Review, Vol 3, Nos. 2 & 3, p. 237-252, 2001
- [3] H. Kerzner, *Project Management a Systems Approach to Planning, Scheduling, and Controlling*, Tenth Edition, John Wiley & Sons, 2009
- [4] B. Pop, *Building an Automated Task Delegation Algorithm for Project Management and Deploying It As Saas*, Studia Univ. Babeş-Bolyai, Informatica, Volume LVIII, Number 2, June, 2013
- [5] B. Pop, F. Boian, *Comparative Study of Task Delegation Models in Software As a Service Project Management Applications*, Knowledge Engineering: Principles and Techniques Conference, KEPT, Cluj-Napoca, July 5-7, 2013
- [6] <http://cassandra.apache.org>
- [7] <http://automated.pm>