A JOINT OPTIMIZATION ALGORITHM FOR DISPATCHING TASKS IN AGENT-BASED WORKFLOW MANAGEMENT SYSTEMS

Pavlos Delias, Anastasios Doulamis and Nikolaos Matsatsinis
Dept. of Production Engineering and Management, Technical University of Crete, University Campus, Chania, Greece

Keywords: Task allocation, agent-based workflow, workflow management systems, joint optimization.

Abstract: Workflow problems generally require the coordination of many workers; machines and computers. Agents provide a natural mechanism for modelling a system where multiple actors operate, but they do not explicitly support coordination schemes. Efficient task allocation to these actors is a fundamental coordination prerequisite. A competent allocation policy should address both system performance issues and users' quality demands. Since these factors are often contradictory, an efficient solution is hard to be identified. In this study, we suggest a task delegation strategy that jointly optimizes system performance (as expressed by workload balancing) and quality demands (as expressed by minimum task overlapping). A consistent modelling approach allows us to transform data of both these factors into a matrix format. The next step is to exploit the Ky-fan theorem and the notion of generalized eigenvalues to optimally solve the task allocation problem. A simple scheduling policy and an experimental setup were applied to test the efficiency of the proposed algorithm.

1 INTRODUCTION

Workflow Management Systems (WfMS) manage business processes through Process Definitions (WfMC, 1999). A process definition consists of a network of activities, while it specifies not only which tasks are performed and their order, but also those who perform them as well. In fact, what is commonly specified is the performer's organization role rather than the actual actor. This information is to be defined by the process modeller during the build time. Yet, during run-time, the actual actor who will finally be in charge of the activity must be decided. Such decisions are configured by the *task allocation policies* of the system and their execution is a critical workflow enactment service. It is through a task allocation policy that an initial order on workflow participants is imposed and that process instances are executed in accordance with their definitions.

An efficient task allocation policy should address performance criteria (such as throughput; computing speed, etc.) and quality criteria, as being defined by the customers. Three general evaluation criteria are proposed to assess the allocation decision, Quality; Cost and Time (O'Brien & Wiegand, 1998). A slightly different set of criteria replaces the "Quality" with "Constraints" and "Success" (Debenham, 2002). Although these criteria outline a comprehensive framework, their interpretation into measurable factors could be rather ambiguous. To overcome this ambiguity, (O'Brien & Wiegand, 1998) and (Debenham, 2002) use an agent negotiation context. The incorporation of Service Level Agreements (SLA's) to bind the negotiation process is an intuitive way to quantify the evaluation criteria (Jennings, Norman, Faratin, O'Brien, & Odgers, 2000; Shepherdson, Thompson, & Odgers, 1999).

A different approach concerning agent-based WfMS is to use a hierarchical structure to dispatch tasks. A central entity is responsible to decide an allocation plan that is later notified to the task executors. The coordination agent of (Madhusudan, 2005) is enabled to dynamically allocate tasks; while a central "Judging Machine" matches agents and work items in (Qiu, Wang, & He, 2005). Manager agents may also assign task as one of their ordinary duties (Aye & Tun, 2005; Manmin & Huaicheng, 1999). A special decision-making agent in (Xu, Qiu,

Delias P., Doulamis A. and Matsatsinis N. (2008).
A JOINT OPTIMIZATION ALGORITHM FOR DISPATCHING TASKS IN AGENT-BASED WORKFLOW MANAGEMENT SYSTEMS.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - AIDSS*, pages 199-206
DOI: 10.5220/0001688601990206

& Xu, 2003) combines data from its peer agents according to the business rules; workflow specification; and workflow process definition to give a list of optimal advices for its own organization. In (Cao, Wang, Zhang, & Li, 2004) the assignment decision (service calling) is based on services' QoS.

Following (Zur Muehlen, 2004), the property that classifies the above methodologies into different categories is with respect to their synchronization mechanism. According to his classification scheme, workflow coordination is a property of synchronization, and *three alternatives for coordinating an allocation policy can be identified*: the hierarchical mode; the negotiation mode and the scheduling-oriented one. Methods of the first category are not scalable as they are prone to bottlenecks, while methods of the second category can operate only in a fully automated environment.

In this paper, we will advocate that a methodology of the third category (a scheduling-oriented one) can overcome these pitfalls, while providing an efficient and optimized allocation scheme. Most of the methods proposed for scheduling-oriented allocation in agent-based WfMS exploit simplified techniques, such as the First-Come-First-Served (FCFS); Earlier-Deadline-First (EDF); and Least-Laxity-First (LLF). However, the aforementioned schemes determine the order in which the tasks are considered for assignment to agents and they cannot be used to determine the specific agent on which the selected tasks are assigned to. Besides, these rules do not optimize the user's quality requirements as far as the tasks are concerned (e.g., tasks' deadlines).

User's requirements, however, are often contradictory. Meeting the requirements of one user should not be achieved by sacrificing the requirements of another user. When the desired users' requirements cannot be fulfilled, we should assign the tasks to the available agents in order to *minimize* degradation of the user's quality demands.

On the other hand, a successful scheduling scheme should *maximize* the overall system performance (i.e., workload balancing) so that the system design is not wasteful. To the authors' knowledge, there is not any optimal strategy which dispatches tasks to the available agents in a way that the a) degradation of user's quality requirements is minimized while simultaneously b) the workload optimally balanced. In this paper, such an optimal strategy is proposed exploiting concepts from the Ky-Fan theorem (Nakic & Veselic, 2003). The Ky-Fan theorem states that the two above mentioned criteria can be solved by the use of a generalized eigenvalue problem.

The optimization criteria and the modelling of our approach are presented in the next section. In section 3 & 4 we present the proposed algorithm and discuss its efficiency. Experimental results and further discussion conclude the paper.

2 MODELLING APPROACH

2.1 The Task Dispatching Process

We adopt the concept and terminology of WfMC (WfMC, 1999) that defines a business process as a network of atomic activities. Each activity is a logical piece of work that can be executed individually by one actor. In this paper, the terms "activity" and "task" will be used interchangeably. Since a process is a directed graph, each activity may or may not have dependencies with other ones. These dependencies are explicitly described in the process definition, thus they are considered a priori known.

We consider a Time Window T when P processes demand for execution. This time window can be considered as a time interval after which a new dispatching procedure is activated. The system can decompose these processes into atomic tasks through the available process definitions, so it is equivalent to say that a set of tasks demands for execution. Let us denote these tasks as T_i^p, i=1,2,..,N, $p=1,2,...,P$. Variable N denotes the number of tasks, belonging to P process instances, that ultimately request to be executed by an agent in the system. An agent may be either a software entity or a human. In any case, they must hold the following critical assumption: *all agents are capable of executing all tasks.* This assumption is necessary for letting us focus on the scheduling-orientation of the proposed allocation scheme. In a later section, we discuss how this limitation can be tackled.

Let us also denote as ST^p the desired *Start Time* for the process instance p and as FT^p the desired *Finish Time* for the respective process instance. Activities' *Start Times* and *Finish Times* are denoted similarly as ST_i^p and FT_i^p respectively. Assuming that we know (or we can estimate) the execution duration d_i of each activity and the *Start Time* for every process, the system can easily calculate the *Start Time* and the *Finish Time* for every task in the system. For the sake of simplicity, we may ignore

the *p* index of the task notation for the rest of the paper, as we address the allocation issue globally and not respective to a process-level.

In this paper, we assume that the tasks are assigned in a *non-preemptable, non-interruptible* way. A task is said to be non-preemptable if once it begins execution by an agent, it has to be completed by that agent. Additionally, a task is said to be *non-interruptible* if once it starts execution it cannot be interrupted by other tasks and resume execution later. Under this assumption, once a task has been assigned to an agent for execution and another task requests for service during the execution time interval, then, the latter task should be assigned either to another agent (which is not reserved at the requested time interval) or undergo violation of its quality requirement, i.e., its deadline. To prevent this from happening, we define as z_{ij} the non-overlapping measure between tasks T_i and T_j (Tasks T_i and T_j may or may not belong to the same process instance). Since non-overlapping is the desired situation, we define z_{ij} as

$$z_{ij} = \begin{cases} a & T_i, T_j \text{ non-overlapped} \\ 0 & T_i, T_j \text{ overlapped} \end{cases} \quad (1)$$

where $\alpha > 0$ any positive non-zero value.

Finally, we need to denote as A_m the set of all tasks executed by the m^{th} agent. Sets A_m, for different agents *m, m=1,2,…,M,* are mutually exclusive, meaning that a task cannot be split and executed collectively by different agents, assuming a non-interruptible scheduling scenario.

2.2 Optimization Criteria

Recalling from section 1, an efficient allocation policy is the one that maximizes i) the percentage of the active agents (optimizes the workload balancing) while ii) simultaneously minimizes the distortion of the tasks' quality requirements. The first condition is of critical importance for the system performance, since, otherwise, resources are wasted (agent idleness) or not properly used (task overloading). The second condition states that the allocation policy should respect user's quality parameters as much as possible. We evaluate violation of deadlines and non-dedicated execution of tasks as quality metrics. When an agent executes at the same time more than one activity, it will inevitably split his capacity across the activities. This will lead to broken deadlines and potentially to reduced quality of the deliverable.

Based on the above mentioned requirements, we infer two optimization criteria: a) Workload balancing as the minimization of the non-overlapping measure among tasks of different agents and b) Quality of Service (QoS) as the maximization of the same non-overlapping measure among all the tasks dispatched to a specific agent. Using equation (1), one can express the non-overlapping degree among tasks of different agents as the sum of the non-overlapping degrees of all tasks assigned to the m^{th} agent with the rest ones, normalized over the sum of non-overlapping degrees between tasks in the m^{th} and all tasks, pending in the system. The corresponding equation is:

$$W_m = \frac{\sum_{i \in A_m, j \notin A_m} z_{ij}}{\sum_{i \in A_m, j \in V} z_{ij}} \quad (2)$$

where *V* is the set of the pending tasks.

Low values of W_m mean that many other agents in the system are concurrently active with the m^{th} agent. On the contrary, as W_m increases, the number of concurrently active agents with the m^{th} one decreases. In the same way, we can express QoS as:

$$Q_m = \frac{\sum_{i \in A_m, j \in A_m} z_{ij}}{\sum_{i \in A_m, j \in V} z_{ij}} \quad (3)$$

The numerator of (3) expresses the sum of the non-overlapping degrees for all tasks of the m^{th} agent. The denominator of equations (2) and (3) expresses the non-overlapping values of the tasks executed by agent *m* with all the *N* tasks including the ones that are executed by the m^{th}. The denominator is used in (2) and (3) for normalization purposes. Instead, optimizing only the numerator of (3) would favour the trivial solution of one task per processor. The Q_m expresses a measure of the overall QoS violation for the tasks' assigned to the m^{th} agent. As Q_m increases, tasks' overlapping, thus QoS violation decreases for the m^{th} agent.

It is quite straightforward to generalize the above optimization metrics. The overall QoS violation measure will be

$$Q = \sum_{m=1}^{M} Q_m \quad (4)$$

while the global workload balancing index will be

$$W = \sum_{m=1}^{M} W_m \quad (5)$$

The ultimate goal of our allocation policy will be to maximize Q while simultaneously minimize W. Combining equations (2), (3), (4) and (5), we get

$$W + Q = M \quad (6)$$

recalling from section 2.1 that M stands for the number of available agents.

Since M is a constant number, equation (6) means that *maximization* of Q simultaneously yields a minimization of W and vice versa. Hence, in our problem, the two aforementioned optimization requirements are in fact identical and they can be satisfied in parallel. Therefore, it is sufficient to optimize only one of the two criteria. In our case, and without loss of generality, we select to minimize W, estimating an optimal task assignment to the M agents, that is a dispatching policy which minimizes the following equation

$$\hat{A}_m : \min W = \min \sum_{m=1}^{M} \frac{\sum_{i \in A_m, j \notin A_M} z_{ij}}{\sum_{i \in A_m, j \in V} z_{ij}}, \forall m \quad (7)$$

where $\hat{A}_m$, is the estimated set of tasks executed by the m^{th} agent

3 THE TASK DISPATCHING ALGORITHM

Optimization of equation (7) is a NP-complete problem. Even for the toy case of two agents, ($M=2$), the optimization of (7) is practically impossible to be implemented for large number of tasks. However, we can overcome this difficulty by transforming the problem of (7) into a matrix based representation. Then, an approximate solution in the discrete space can be found using concepts derived from eigenvalue analysis.

3.1 Matrix Representation

Let us denote as $\mathbf{Z} = [z_{ij}]$ a matrix which contains the values of the non-overlapping measure z_{ij} for all tasks T_i and T_j. Let us also denote an Nx1 indicator vector $\mathbf{e}_m = [\cdots e_m^u \cdots]^T$ whose elements e_m^u are given by

$$e_m^u = \begin{cases} 1 \text{ if task } T_u \text{ is executed by agent } m \\ 0 \qquad\qquad\qquad \text{Otherwise} \end{cases} \quad (8)$$

The indicator vector $\mathbf{e}_m$ points out which tasks are allocated to whom. M different indicator vectors exist, one per agent. Therefore, the optimization problem of (7) is equivalent to the estimation of the optimal indicators vectors $\hat{\mathbf{e}}_m, \forall m$ which minimize equation (7). Consequently, equation (7) can be written as

$$\hat{\mathbf{e}}_m, \forall m : \min W = \min \sum_{m=1}^{M} \frac{\sum_{i \in A_m, j \notin A_M} z_{ij}}{\sum_{i \in A_m, j \in V} z_{ij}} \quad (9)$$

The main difficulty in (9) is that its right part is *not expressed* as a function of the indicator vectors $\mathbf{e}_m$. Since direct minimization is not straightforward, we need to re-write the right part of equation (9) in a form of vectors $\mathbf{e}_m$. For this reason, let us denote as $\mathbf{L} = diag(\cdots l_i \cdots)$ a diagonal matrix, whose elements l_i, i=1,2,..N express the cumulative non-overlapping degree of the task T_i with all the remaining tasks. That is

$$l_i = \sum_j z_{ij} \quad (10)$$

Using matrices **L** and **Z**, we can express the numerator of (9) as a function of vectors $\mathbf{e}_m$. In particular,

$$\mathbf{e}_m^T (\mathbf{L} - \mathbf{Z}) \mathbf{e}_m = \sum_{i \in A_m, j \notin A_m} z_{ij} \quad (11)$$

In a similar way, the denominator of (9) is related with the indicator vector $\mathbf{e}_m$ as follows

$$\mathbf{e}_m^T \mathbf{L} \mathbf{e}_m = \sum_{i \in A_m, j \in V} z_{ij} \quad (12)$$

Using (11) and (12), we can re-write (9) as

$$\hat{\mathbf{e}}_m, \forall m : \min W = \min \sum_{m=1}^{M} \frac{\mathbf{e}_m^T (\mathbf{L} - \mathbf{Z}) \mathbf{e}_m}{\mathbf{e}_m^T \mathbf{L} \mathbf{e}_m} \quad (13)$$

3.2 Optimization in the Continuous Domain

Assuming *non-interruptible* tasks, we allow agents either to undertake the whole task; or let another agent do the work. That means that the $\mathbf{e}_m$ vectors take binary values (1 for assignment, 0 otherwise). In other words, we can form the indicator matrix

$\mathbf{E} = [\mathbf{e_1} \cdots \mathbf{e_M}]$, the columns of which refer to the M system agents, while the rows to the N tasks. Then, the rows of **E** have only one value equal to one while all the rest values are zero. Optimization of (13) under the binary representation of the indicator matrix **E** is still a NP hard problem. However, if we relax the indicator matrix **E** to take values in continuous domain, then we can solve the problem in polynomial time. We call $\mathbf{E}_M$ the *relaxed version of the indicator matrix* **E**. The elements of the relaxed matrix take real values.

It can be proven that in the continuous domain the right part of (13) can be written as

$$W = M - trace(\mathbf{Y}^T \mathbf{L}^{-1/2} \mathbf{Z}\, \mathbf{L}^{-1/2} \mathbf{Y}) \quad (14)$$

Subject to

$$\mathbf{Y}^T \mathbf{Y} = \mathbf{I} \quad (15)$$

where **Y** is a matrix which is related with the matrix $\mathbf{E}_M$ through the following equation

$$\mathbf{L}^{-1/2} \mathbf{Y} = \mathbf{E}_M \mathbf{\Lambda} \quad (16)$$

and $\mathbf{\Lambda}$ any $M \times M$ matrix. In this paper, we select $\mathbf{\Lambda}$ to be equal to the identity matrix, $\mathbf{\Lambda}=\mathbf{I}$. Then, the relaxed indicator matrix $\mathbf{E}_M$, which is actually the matrix we are looking for, is calculated as

$$\mathbf{E}_M = \mathbf{L}^{-1/2} \mathbf{Y} \quad (17)$$

Minimization of the problem (14)-(15) is obtained through the Ky-Fan theorem (Nakic & Veselic, 2003). The Ky-Fan theorem states that the maximum value of the $trace(\mathbf{Y}^T \mathbf{L}^{-1/2} \mathbf{Z}\, \mathbf{L}^{-1/2} \mathbf{Y})$ subject to the constraint of $\mathbf{Y}^T \mathbf{Y} = \mathbf{I}$ is equal to the sum of the M ($M<N$) *largest eigenvalues of matrix* $\mathbf{L}^{-1/2} \mathbf{Z}\, \mathbf{L}^{-1/2}$. Consequently,

$$\max\{trace(\mathbf{Y}^T \mathbf{L}^{-1/2} \mathbf{Z}\, \mathbf{L}^{-1/2} \mathbf{Y})\} = \sum_{i=1}^{M} \lambda_i \quad (18)$$

where λ_i refers to the i^{th} large eigenvalue of matrix $\mathbf{L}^{-1/2} \mathbf{Z}\, \mathbf{L}^{-1/2}$. However, maximization of (18) leads to minimization of W in (14). Thus, it is clear that the *minimum value* of W will be

$$\min W = M - \sum_{i=1}^{M} \lambda_i \quad (19)$$

The Ky-fan Theorem also states that this minimum value of W [equation (19)] is obtained through the matrix

$$\mathbf{Y} = \mathbf{U} \cdot \mathbf{R} \quad (20)$$

where **U** is a $N \times M$ matrix the columns of which are the *eigenvectors* of the M largest eigenvalues of matrix $\mathbf{L}^{-1/2} \mathbf{Z}\, \mathbf{L}^{-1/2}$ and **R** an *arbitrarily rotation matrix* (i.e., orthogonal with determinant of one). Again, a simple approach is to select matrix **R** as the identity matrix, i.e., **R**=**I**, so

$$\mathbf{Y} = \mathbf{U} \quad (21)$$

Finally, we calculate the optimal relaxed indicator matrix $\hat{\mathbf{E}}_M$ in the continuous domain as

$$\hat{\mathbf{E}}_M = \mathbf{L}^{-1/2} \mathbf{U} \quad (22)$$

3.3 Discrete Approximation

The optimal matrix $\hat{\mathbf{E}}_M$ of (22) has not the form of the indicator matrix **E** since its values are continuous, while the elements of **E** [see (13)] are binary. We recall that binary values are the desired ones since we have assumed a *non-interruptible, non-preemptable scheduling policy*. Consequently, in order to accept the optimal solution of (22) as a solution for our problem, we have to round the continuous values of $\hat{\mathbf{E}}_M$ in a discrete form that approximate matrix **E**.

One simple solution, regarding the rounding process, is to set the maximum value of each row of matrix $\hat{\mathbf{E}}_M$ to be equal to 1 and let the remaining values to be zeros. However, such an approach yields unsatisfactory performance in case that there is not any dominant maximum value at every row of $\hat{\mathbf{E}}_M$. Furthermore, it handles the rounding process as N independent problems, implying that each task is delegated without regarding the allocation of the others. An alternative approach, which is adopted in this paper, is to treat the N rows of matrix $\hat{\mathbf{E}}_M$ as M-dimensional feature vectors. Each one of these feature vectors indicates the association degree of each task and the respective m^{th} system's agent.

More specifically, after we have normalized the rows of $\hat{\mathbf{E}}_M$, we apply the k-means clustering algorithm, considering the rows of $\hat{\mathbf{E}}_M$ as the population to be clustered in M classes. The k-means algorithm comprises three phases, the initialization; the clustering construction; and the updating phase.

Initialization: In this phase, the algorithm arbitrarily selects a set of $\hat{\mathbf{E}}_M$'s rows as centers of the classes that are to be constructed. The number of selected rows equals M. That means that each class will contain the tasks assigned to an agent.

Clustering Construction: In this phase, the remaining rows of $\hat{\mathbf{E}}_M$ are clustered to the M classes using a metric distance. In particular, a row

(namely a task) is assigned to a class by comparing its vector with the class centers and selecting as the appropriate class, the one with the most proximate center.

Updating: After the classification, new centers are created as the means of all vectors belonging to a class. In case that these centers are different from the previous ones, a new process takes place and the algorithm moves on to the clustering construction phase for further processing. On the contrary, if the new centers are exactly the same with the previous ones, meaning that the same task assignment have been concluded, no further processing is required and the clustering is terminated.

The performance of the k-means algorithm highly depends on the initial selection of the class centers, even though it can be proven that the k-means always converges to a solution. Thus, the effectiveness of the dispatching policy is actually influenced by the selection of the initial matrix rows. In this paper, to overcome such a drawback and simultaneously to search for new possible solutions that will yield, in relatively small time, a satisfactory approximation of the optimal solution in the discrete domain, we repeat the experiment by selecting each time different rows for the initialization, which in turn, will provide different solutions. Among all selections, the minimum is returned as the finest approximation.

4 ALGORITHM EFFICIENCY

We define the task arrival rate λ as the number of tasks, say N, requesting for execution within a time window T (see Section 2). We shall also notice that task arrival rate λ is calculated through the process arrival rate λ'. Finally, task arrival rate will be

$$\lambda = \frac{N}{T} \tag{23}$$

while the task average duration will be

$$D = \sum_{i=1}^{N} d_i \Big/ N \tag{24}$$

An important aspect which determines dispatching efficiency is the task granularity g, measured as the ratio of the average task duration D over the time window T.

$$g = \frac{D}{T} \tag{25}$$

It is expected that as granularity increases, dispatching performance decreases since more execution capacity is required. Given a granularity g and a rate λ, the *lower bound* of available agents required for achieving the two goals (workload balancing and QoS) is the *lower bound* of available agents required for achieving zero task overlapping. This bound is

$$B = \frac{ND}{T} = N \cdot g \le M_{opt} \tag{26}$$

where M_{opt} refers to the minimum number of agents required for achieving no task overlapping under an exhaustive search allocation scenario. It should be mentioned that M_{opt} cannot be reached in real life scenarios, since the exhaustive search algorithm is a NP-hard problem. The lower bound of (26) is achieved in the extreme case when the tasks arrive one right after the other, while their durations cover every gap within the time window T.

Given the lower bound B of the agents required so that no tasks' overlapping is encountered, we can define the allocation efficiency as

$$e(S) = \frac{B}{M(S)} \tag{27}$$

where S refers to the algorithmic strategy adopted to approximate the exhaustive search policy, $M(S)$ the minimum number of agents estimated through the algorithm S and $e(S)$ the respective algorithm's efficiency. Using equation (26), it is clear that

$$e(S) \le \frac{M_{opt}}{M(S)} \tag{28}$$

As a result of (26), the lower bound B does not take integer values. Since, however, B expresses the minimum bound of the required individual agents; the real values of B should be rounded to the next integer so that they are strictly greater than B, that is

$$\varepsilon(S) = \frac{\lceil B \rceil}{M(S)} \tag{29}$$

where as $\lceil \cdot \rceil$ we indicate the ceil operator and as $\varepsilon(S)$ the rounded efficiency for the algorithm S.

5 EXPERIMENTAL RESULTS

5.1 A Verifying Scheduling Policy

In order to test our algorithm we apply the following workflow scenario: A central, manager agent receives a batch of process instances that demand for execution. The manager agent has access to the process definition repository, so it can decompose the processes into atomic tasks. As described in

Section 2.1, the manager agent can calculate the *Start Time* and *Finish Time* for every task. It also maintains a list of available agents. This list can be dynamically updated, through direct communication among agents or through brokers as discussed in section 6. The results of the proposed dispatching policy are communicated to the agents, who are committed to execute the assigned tasks.

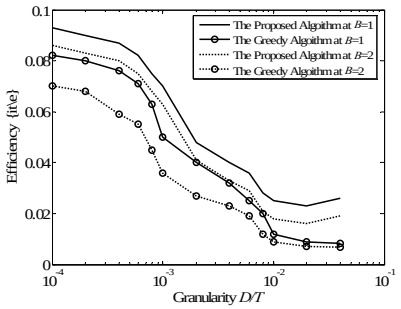

Figure 1: Efficiency versus granularity for different tasks loads, (B values) for the proposed algorithm and the greedy one.

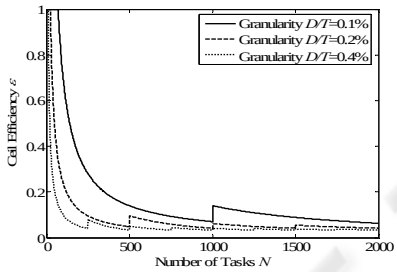

Figure 2: Ceil Efficiency ε versus the number of tasks for different granularity values.

Before starting task execution, every agent checks its assigned task for dependencies. If there are any precedence requirements, the agent must assure that the precedent tasks have indeed been accomplished. For this reason, we employ a special field in the agents' architecture: a check board. A check board is actually a binary vector whose length equals the number of tasks assigned to the agent. Thus, check board elements are either 1 if the agent has successfully completed the task; or 0 otherwise. Since agents are notified about the allocation scheme, they know which agent is in charge of the precedents task, so they can directly query it. In case, that a task has one or more precedents, it can not be started, unless all the precedents are accomplished.

In case of a task failure, the agent not only put a zero in the corresponding place at its check board, but it also informs the manager agent. The latter, cancels all the tasks that belong to the same instance and logs the process failure. This instance has to be assigned again during the next dispatching procedure. However, exception handling is a major issue that needs to be addressed more efficiently, yet it is out of the scope of this paper.

5.2 Experimental Setup

A process definition generator is created to provide us with simulation data. Every generated definition comprises a random number of activities, split into four blocks: A sequence, a parallel block, an OR gateway and a sequence again. Process instances and activities duration times are also randomly generated. We compare our approach with a *greedy* one, which selects a locally optimum choice for every task. In particular, the *greedy* algorithm assigns each task to an agent so that no task overlapping is encountered, by exploiting the current local load of each agent. If all agents are loaded with tasks, then the pending tasks undergo violation of their QoS requirements.

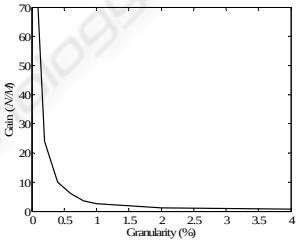

Figure 3: Gain defined as the ratio of N/M versus granularity.

5.3 Simulation Results

Figure 1 shows the efficiency measure e versus the variation of the granularity value D/T [see(27)] for two algorithmic strategies: the proposed one and the greedy approach. As is observed, the efficiency increases as granularity decreases, meaning that for tasks of relatively short duration compared to the window time better allocation can be achieved in accordance to the tasks' load. However, the improvement is saturated for low values of granularity. Similarly, we observe a reduction of the efficiency e as B increases for the same granularity (i.e., an increase in tasks' load), meaning that more agents are required for task execution to achieve no overlapping. In this Figure, we compare the results derived by the proposed algorithm, with the ones stem from the greedy scheduling approach. For all values, the proposed scheme outperforms the greedy approach, meaning that less agents are required to

efficiently schedule the same number of workflows using the proposed scheme than the greedy one.

Figure 2 shows the discrete efficiency value denoted as ε in (29) versus the number of tasks. The efficiency decreases, however, peaks are presented with a periodic order stemming from round function in (29).

Finally, the gain, defined as the ratio of the number of tasks over the minimum number of agents required for achieving no overlapping versus granularity is shown in Figure 3. It is clear that, the gain is exponentially increases for low values of granularity, instead of the high ones.

6 DISCUSSION

In this paper we propose a scheduling-oriented task dispatching policy. The application context of the suggested algorithm is WfMS where agents operate. A major assumption that guided our approach is that all agents are capable of executing all tasks (see section 2.1). Apparently, this is not always the real situation. To overcome this limitation, we may integrate into the dispatching procedure a negotiation step. During that step the manager agent may call for bids and agents that fulfill the capacity requirements may answer. Then the manager agent can decide the allocation plan based on the available agents. Another way is to incorporate broker agents. Broker agents know the capacities of their teams, so they could be in charge for the negotiation process. Alternatively, the manager agent could dispatch tasks to brokers, considering them as similar entities, who in turn, will allocate their tasks to their agents.

Concluding, as simulations results demonstrate, the task dispatching policy that we propose can efficiently optimize both system's performance and user's QoS requirements. Adopting optimization criteria based on measures of task overlapping, we approximate an NP-complete problem with an algorithm of polynomial order. The results of our algorithm can feed the workflow engines of a WfMS system and allow them an adequate task allocation.

ACKNOWLEDGEMENTS

This work is supported by 03ED375 PENED project, co-financed by 75% from E.U and 25% from GRST.

REFERENCES

Aye, T., & Tun, K. M. L. (2005). *A Collaborative Mobile Agent-based Workflow System.* Paper presented at the 6th Asia-Pacific Symposium on Information and Telecommunication Technologies, 2005. APSITT 2005 Yangon, Myanmar.

Cao, J., Wang, J., Zhang, S., & Li, M. (2004). A dynamically reconfigurable system based on workflow and service agents. *Engineering Applications of Artificial Intelligence, 17*(7), 771-782.

Debenham, J. (2002). *Who does what in a multiagent system for emergent process management.* Paper presented at the Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS' 02), Lund, Sweden.

Jennings, N. R., Norman, T. J., Faratin, P., O'Brien, P., & Odgers, B. (2000). Autonomous Agents For Business Process Management. *Applied Artificial Intelligence, 14*(2), 145-189.

Madhusudan, T. (2005). An agent-based approach for coordinating product design workflows. *Computers in Industry, 56*(3), 235-259.

Manmin, X., & Huaicheng, L. (1999). *Cooperative software agents for workflow management system.* Paper presented at the Fifth Asia-Pacific Conference On Communications and Fourth Optoelectronics and Communications Conference APCC/OECC '99, Beijing, China.

Nakic, I., & Veselic, K. (2003). Wielandt and Ky-Fan Theorem for Matrix Pairs. *Linear Algebra and its Applications, 369*(17), 77-73.

O'Brien, P. D., & Wiegand, M. E. (1998). Agent based process management: applying intelligent agents to workflow. *The Knowledge Engineering Review, 13*(2), 161-174.

Qiu, J., Wang, C., & He, Y. (2005). *Research on application of intelligent agents in the workflow management system.* Paper presented at the 2005 IEEE Networking, Sensing and Control, ICNSC2005, Tucson, Arizona, USA.

Shepherdson, J. W., Thompson, S. G., & Odgers, B. R. (1999). Decentralised Workflows and Software Agents. *BT Technology Journal, 17*(4), 65-71.

Workflow Management Coalition. (1999). *Terminology & Glossary* (WfMC Specification documents No. WFMC-TC-1011).

Xu, Q., Qiu, R., & Xu, F. (2003, 5-8 Oct. 2003). *Agent-based workflow approach to the design and development of cross-enterprise information systems.* Paper presented at the IEEE International Conference on Systems, Man and Cybernetics, 2003. , Washington, D.C., USA.

Zur Muehlen, M. (2004). Organizational Management in Workflow Applications – Issues and Perspectives. *Information Technology and Management, 5*(3), 271-291.