ODDMS: Online Distributed Dynamic Meeting Scheduler

Archana Nigam and Sanjay Srivastava

Abstract With advent of social media, need for scheduling very large meetings while proving a degree of privacy to the participants has become an important problem. Existing solutions based on a global calendar expose individuals data to the calendar provider and thus are unsuitable for open meetings with quorum constraints. We propose an online distributed dynamic meeting scheduler (ODDMS). It is able to efficiently schedule meetings involving a large number of participants, without having complete knowledge of individual participants and their preferences, thus preserving privacy. The algorithm uses a modified negotiation-based distributed schedule that resolves the problem of deadlock and contention using hidden naive Bayes learning method. We compare our work with a baseline centralized algorithm and two existing algorithms based on voting mechanism and naive Bayesian methods. Simulation studies show that ODDMS performs similar to baseline centralized algorithm under light load condition and significantly outperforms the existing distributed algorithms under heavy load condition.

Keywords Distributed system · Meeting scheduling · Knowledge base · Deadlock handling · Hidden naive Bayes · Contention avoidance

1 Introduction

Scheduling a meeting is a naturally distributed task which can be defined as a method of finding the suitable and optimal time slot based on preferences and priorities of both host and participants [6]. If participants involves are huge in number, i.e., in thousands, for example, social media-based meeting, alumni meetup,

A. Nigam (✉) · S. Srivastava
Dhirubhai Ambani Institute of Information and Communication Technology, Gandhinagar, India
e-mail: archana_nigam@daiict.ac.in

S. Srivastava
e-mail: sanjay_srivastava@daiict.ac.in

S. Fong et al. (eds.), *ICT Analysis and Applications*, Lecture Notes in Networks and Systems 93, https://doi.org/10.1007/978-981-15-0630-7_19

event schedules, etc., then such problem involves more complexity, as many meeting being scheduled at the same time considering several constraints [8]. Till now most of the work had been done for organization-level meetings; this motivates us for a more realistic case where meeting scheduling for non-organizational setup is involved. Realistic distributed meeting scheduling is an online problem in which a new meeting arrives while the number of the meeting is being scheduled concurrently; therefore, the status of the local and non-local resources may change over time [13]. In this paper, we are considering a multi-agent environment. There are two types of agents in our system: host agent (HA) and participant agent (PA). HA is responsible for scheduling the meeting on the users behalf. PA responds on behalf of users who are in the guest list of the meeting. In centralized meeting scheduling, a central host has all the information regarding preference and availability of participants; therefore, the host takes a decision and schedules a meeting on their behalf. However, this method breaches the privacy constraint as each participant and central host can see the data of other participants and also participants remain isolated from decision-making process. Whereas in distributed meeting scheduling, information is kept private by sharing the limited information among entities which are involved in a meeting; therefore, entities have partial information about each other [2]. This information transfer is taken place through negotiation where an agent negotiates with other agents to select a common schedule considering the user's preferences and calendar. Another approach in distributed meeting scheduling is a voting-based method, HA releases a set of date, and all the PAs are asked to adapt date set according to their constraint and rank the date set accordingly. Negotiation-based method gives more freedom to the individual participant to take part in decision-making. In the distributed system because of partial information, meeting scheduling requires a large number of negotiation rounds and to make it converge learning method is adopted. In an online distributed system, if negotiation will not converge in every round, then with an increase in a number of pending meeting requests deadlock condition may arise. In this paper, we are solving a contention and deadlock problem in distributed meeting scheduling system using a hidden naive Bayes learning method. In this paper, we propose an online distributed dynamic meeting scheduler (ODDMS) which efficiently schedules meetings involving a large number of participants, without having complete knowledge of individual participants preferences, thus preserving privacy. To make our system more realistic, we introduce two important factors in our system: quorum and overlapped criteria. Quorum is defined as the number of participants mandatory for the meeting to be held, and overlapped criteria are defined as the number of participants common in more than one meeting. Quorum is important as it signifies the minimum number of participants to successfully schedule a meeting, failing which meeting cannot be scheduled. We compare ODDMS with techniques found in literature and the result shows that ODDMS perform better than most of the distributed meeting scheduling system. As per best of our knowledge under this setting, no work has been done in the literature. The paper is organized as follows: In Sect. 2, we discuss the literature review. In Sect. 3, the problem statement is given. Section 4 discusses the proposed architecture and algorithm. Section 5 discusses simulation results; we conclude in Sect. 6.

2 Literature Survey

Doodle [5] is a well-known service that allows users to find all common availability. However, it does not provide privacy as each user and doodle server can see the busy or free state of every other user. To solve the distributed meeting scheduling problem, two types of approaches were discussed in the literature: negotiation-based and voting-based. In [10], the author solves distributed meeting scheduling problem using negotiation technique, where a host agent communicate with all other agents in order to schedule a meeting and feasible schedule is searched based on participants calendar information passed during negotiation. This approach does not takes into account participant's preferences, whereas in [11] participant's preferences and calendar information is communicated during negotiation, therefore, privacy is not warranted. The main objective of using a learning method in distributed meeting scheduling is to reduce the burden of negotiation by calculating the probability of acceptance of the meeting based on past knowledge [7]. In [7], the author uses naive Bayes classifier for calculating the probability of meeting acceptance. But the naive Bayes classifier works for independent attributes, i.e., it considers only participants availability, not considering other attributes such as preferences and priority. Also, it does not use negotiation result for knowledge updating. In [13], Bayesian network is used for learning users' preference and this system learn from negotiation result. But learning an optimal Bayesian network has been proved to be NP-hard [3]. In fact, the most time-consuming step in learning a Bayesian network is learning the structure [12].

Another approach in distributed meeting scheduling is voting-based meeting scheduling system. In [2], host transfers a set of dates based on user preferences and priority, and all the participants are asked to adapt the date set according to their constraint and rank the date set accordingly. Date with a high number of votes is chosen as a date for a meeting. Here only date attribute is taken into consideration, other attributes such as time, day, duration are also important. Other concern with negotiation-based and voting-based methods is that they will suffer in the highly loaded scenario where a number of participants are common in more than one meeting for the same time interval and if a quorum is also high. In online distributed system, if negotiation will not converge in every round then with increase in number of pending meeting requests deadlock condition may arise.

3 Problem Statement

A meeting schedule is defined as a set of meetings for a group of participants. Given a set of n meetings and j participants, a scheduling problem is represented as $MS = (A, M)$, where $A = \{1, 2, \ldots, j\}$ is the set of participant and $M = \{m_1, m_2, \ldots, m_n\}$ is the set of n meetings to be scheduled. A time slot is represented as a day, date,

hour tuple $<Da, Dt, H>$. A set of contiguous time slots is called a time interval. A meeting is represented by a tuple:

$$m_u = (A, h, dr, q, \tau)$$

where A is a set of participants of the meeting; h is host agent of the meeting; dr is the duration of the meeting in hours; and q is the quorum value.

τ is the time interval for which the meeting m_u is finally scheduled and is represented by an ordered set $\{<Da_u, Dt_u, H_u>, <Da_u, Dt_u, H_u + 1>, \ldots, <Da_u, Dt_u, H_u + dr_u - 1>\}$ (here Da_u gives the day, Dt_u gives the date, and H_u gives the starting hour for which meeting m_u is scheduled) if the meeting could be scheduled, and by null set otherwise.

3.1 Criteria for System Evaluation

In our system if multiple meeting requests arrive for a same time interval where participants are common and quorum is high, then meeting scheduling will be difficult. Therefore, offered load in our system is defined as

$$L = \begin{cases} L = \widetilde{Q}\widetilde{O}, & \text{if } \tau_i = 1 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

If n meeting request arrives correspond to ith time interval, $\widetilde{Q}$ and $\widetilde{O}$ are the average quorum and overlapped value, respectively, for n meetings. If there is no overlap then the load is null.

We evaluate our system in terms of two first-order metrics, namely, efficiency and satisfaction index. Efficiency reflects the degree to which system succeeds in scheduling the important meetings. Satisfaction index reflects how much satisfy the participants are corresponding to a particular schedule.

Efficiency is defined by the following equation:

$$\eta_i = \frac{|m|}{|M|} \tag{2}$$

where M is total meeting request arrived; m is meeting finally scheduled in i time interval.

Satisfaction index (SI) is defined as a meeting that is scheduled in the first round itself corresponds to optimal time interval choice. Each round of negotiation needed to schedule a meeting comes at a cost of lowered satisfaction. Thus, we define SI for meeting in a time interval as a weighted average, with weight $w = \frac{1}{r}$.

$$\mathrm{SI}_i = \frac{1}{|m|} \sum_{i=1}^{r_{\max}} \frac{|mr_i|}{i}, \qquad |m| = \sum mr_i \tag{3}$$

$r_{\max}$ is the maximum number of negotiation round required for meeting to schedule, $|mr|$ is the number of meeting scheduled in each negotiation round.

If there is total k, then $\widetilde{SI}$ is the average SI, and $\widetilde{\eta}$ is the average fraction of meeting schedule for total k time interval. The primary objective function is to maximize efficiency and satisfaction index simultaneously.

4 Online Distributed Dynamic Meeting Scheduler

In our proposed system, we are considering a multi-agent environment. There are two types of agents in our system, host agent (HA) and participant agent (PA). HA is responsible for scheduling the meeting on the user's behalf. PA responds on behalf of users who are in guest list of the meetings. HA calculates the probability of acceptance of a meeting proposal based on past data. If the probability is high, the proposal is sent to all the PAs. PA can accept the proposal if time interval is free or send a pending message. Finally, if quorum criteria are met, then meeting is confirmed else canceled. At most three rounds of negotiation are involved in our system. Under such setting negotiation-based method works perfectly as compare to voting-based method because negotiation-based method gives more freedom to the individual PA to take part in decision-making. In distributed system due to partial information, proposing a time interval suitable to all is a difficult task and it require large number of negotiation rounds, thus there will be no guarantee that negotiation method will converge. If there are n PA and meeting scheduling require r rounds of negotiation, if r is large then the cost it incurred will be $O(rn)$. To reduce the cost of subsequent round and to make negotiation method converge, learning method is adopted. Therefore to solve this problem, we are using a hidden naive Bayes learning method (HNB) [12]. In online distributed system, if negotiation will not converge in every round then with increase in number of pending requests deadlock condition may arise. Another condition of deadlock which is common in distributed system is hold and wait condition. We are solving the problem of deadlock using weighted priority and time out mechanism. The architecture of both host and participant agents is shown in Fig. 1. In HA architecture, a meeting request is given as input to contention avoidance module, details will be discussed in Sect. 4.1. In contention avoidance module, analysis function calculates the probability of acceptance of meeting request. If probability of acceptance is higher, meeting request is send to all PA. On receiving the message from PA, offer analyser module will check whether quorum criteria are met or not. If met, changes are made into calendar and confirmation is send to PA. On confirmation or rejection of meeting knowledge base is updated. Algorithm 1 describes the working of host agent, and Algorithm 2 describes the working of participant agent. In PA architecture, on receiving a meeting request for the first time

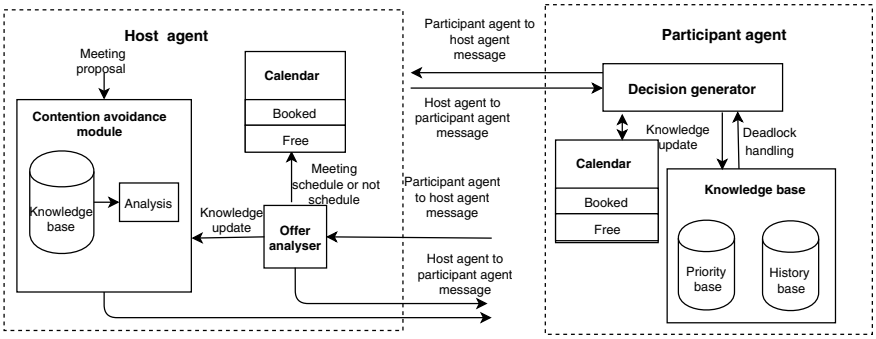


Fig. 1 Host agent and participant agent architecture

decision generator module will check the participant's calendar. If meeting is already booked reject message will be send, if time interval is free but there is request for same time interval from other HA then priority of both the HAs is calculated.

Algorithm 1 Host agent

1: Input: Meeting request
2: **procedure** CONTENTION AVOIDANCE
3: Input: Set D of past history data
4: Output: Probability of success or fail
5: **if** probability of success is high **then**
6: **Send**: meeting proposal to all participant, Call Offer analyser
7: **end if**
8: **end procedure**
9: **procedure** OFFER ANALYSER
10: **if receive**: $accept \geq q$ **then**
11: **Send**: Meeting Confirm;
12: **else**
13: call Meeting Pending;
14: **end if**
15: **end procedure**
16: **procedure** MEETING PENDING
17: **if** $round \leq 2$ **then**
18: negotiation round, call Offer analyser
19: **else**
20: Compute time out, call Offer analyser
21: **end if**
22: **end procedure**

PA send acceptance message to higher priority HA and pending to other. If time interval is free and no conflict, then PA send acceptance to HA. If message is received for the second round then there might be deadlock between two requests for same time interval. Therefore, decision generator performs deadlock handling by calculating

the weighted priority of both the HAs; accept message will be send to one with higher weighted priority and reject to other. If both the HAs have same weighted priority, then PA will send nothing. Both HA will wait for time out. Knowledge base is updated by decision generator and corresponding changes are made into the calendar of participant.

4.1 Contention Avoidance with Hidden Naive Bayes

In distributed system, there is no guarantee that random negotiation method will converge. Also the cost it incurred will be $O(rn)$. Therefore to solve this problem we are using a hidden naive Bayes learning method (HNB) [12]. The advantage of HNB is as it consider various attributes like preferences, priority, availability during learning. Instead of becoming slow when more situation is learned, it is more responsive and improves its precision by learning from negotiation. Therefore, the probability of success in organizing a meeting will increase. In HNB learning method, prior knowledge is combined with observed data to determine the final probability of a hypothesis. It creates a hidden parent for each attribute, which combines the influences from all other attributes [12]. $\{A_1, A_2, \ldots, A_n\}$ are n attributes and C is the class node. The joint distribution represented by an HNB is defined as follows:

Algorithm 2 Participant agent

```
1: procedure DECISION GENERATOR
2:     Input: meeting proposal receive
3:     if round == 1 then
4:         Check calendar availability.
5:         if available and no contention then
6:             send: accepted
7:             if available and contention then
8:                 send: pending, after comparing priority
9:             else
10:                send : reject
11:            end if
12:        end if
13:    end if
14:    if meeting confirm then
15:        Update Calendar
16:    end if
17:    if round == 2 then
18:        call: Deadlock Handling.
19:    end if
20: end procedure
21: procedure DEADLOCK HANDLING
22:    compute weighted priority to break tie
23:    Update History
24: end procedure
```

$$P(A_1, \ldots, A_n, C) = P(C) \prod_{i=1}^{n} P(A_i | A_{hpi}, C) \tag{4}$$

Here in our case class is either success or fail, and attributes are a participant agent, date, time, day, and duration. For example, $P(\text{Agent} = A, \text{date} = 10/5/2019, \text{time} = 10\,\text{am}, \text{day} = \text{Mon}, \text{duration} = 2\,\text{h}, C = \text{success})$. We used the Laplace estimation to avoid the zero-frequency problem [12].

4.2 Deadlock Handling

In online distributed system, if negotiation will not converge in every round then with increase in number of pending meeting requests deadlock condition may arise. Another condition of deadlock which is common in distributed system is hold and wait condition. The problem of deadlock is solved by giving the preference to high priority host agent to break the tie [4]. But in the worst case, if all the host agent has the same priority then deadlock problem will remain the same. Therefore to solve this issue, we use weighted priority between priority list defined by participants' agent as well as priority list derived from the past meeting. To calculate weighted priority we have a smoothing factor β.

$$W_{\text{p}} = \beta p + (1 - \beta)h, \quad 0 \leq \beta \geq 1 \tag{5}$$

where W_{p} is weighted priority, p is priority from priority database, and h is derived priority from history database.

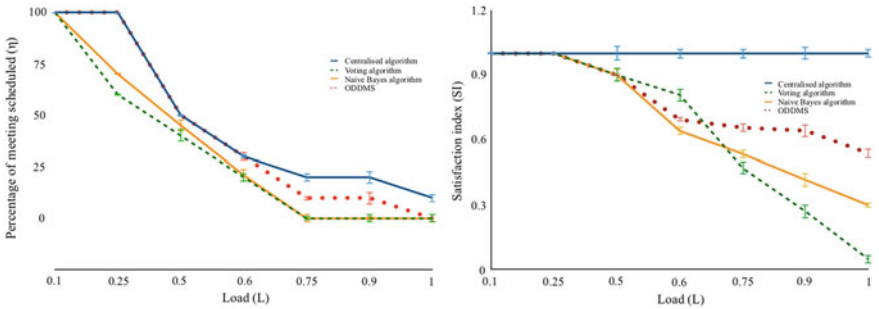
History database. To calculate h, we use naive Bayes classifier [9]. Host agent with high h has done more number of meeting with the participant agent. In worst case if weighted priority is again same for both the host agents, then we use timer to drop particular host agent.

$$T_o = 2 \times \text{RTT}_a + \frac{\text{acc}_i}{q_i} \tag{6}$$

where acc_i is total number of accept received for a meeting i, and q_i is quorum defined for meeting i. RTT_a is the time of accept message reaching from participant agent to host agent. Therefore, we are dropping the host agent whose work done is less compare to other host agent.

5 Simulation Result

In this section, we simulate our design using JADE (Java Agent Development) [1] framework. In our simulation test-bed, we have 1000 meetings spread across 20 time interval; multiple meetings are overlapped randomly in a single time interval.



(a) Percentage of meeting scheduled as a function of offered load

(b) Satisfaction index as a function of offered load

Fig. 2 Performance and satisfaction index as a function of offered load

Each meeting involves number of participants between 500 and 2000. Performance of ODDMS is measured using η and SI as a function of offered load. We compare ODDMS with centralized algorithm, learning-based system [7] and voting-based system [2].

In our simulation results centralized algorithm is used as a benchmark for a comparison, this is the optimal result that can be achieved. In Fig. 2a, we are comparing η as a function of offered load between all the four algorithm. Under low load condition, ODDMS perform similar to centralized algorithm because learning method that we use is more restrictive, so it will send proposal to those who have a high chance of acceptance. Under high load condition, voting-based method gives poor performance because of increase in disagreement which increases the contention in the system and leads to deadlock; similarly in naive Bayes learning algorithm, there is no mechanism to detect and resolve deadlock. Therefore in both of these methods, the number of meeting schedule comes to zero. ODDMS performance is better because we are handling deadlock and contention using learning method. In Fig. 2b, we are comparing the satisfaction index of the four algorithm. In classical centralized algorithm, host has all the preference and priority data of all agents; therefore, it has complete knowledge of when to schedule meeting such that most of the agent will be satisfied. In voting-based method, during moderate-high load condition number of disagreement increases and this disagreement continues till meeting rejected or accepted thus poor satisfaction index. In naive Bayes method, due to contention negotiation round increases which degrade the performance of algorithm. In ODDMS, the number of negotiation round is three; in worst case much less compare to voting and naive Bayes because better scheduling decision is made based on the past knowledge considering all the dependent attributes.

6 Conclusion

We have proposed a distributed algorithm to solve the online meeting scheduling problem. ODDMS is able to efficiently schedule meetings involving large number of participants, without having complete knowledge of individual participants preferences, thus preserving privacy. We introduce the notion of quorum and overlapped criteria. We compare our ODDMS with voting and naive Bayes algorithm. Our simulation result shows that ODDMS is able to schedule about 43% more meeting under low load condition and about 25% under high load condition. Similarly, quality of the scheduled meeting is measured by satisfaction index (SI). SI is about 60% more than the voting and naive Bayes distributed algorithm when offered load reaches to 1.

References

1. Bellifemine F, Caire G, Poggi A, Rimassa G (2003) JADE a white paper. Telecom Ital EXP Mag 3(3). http://www.cis.umassd.edu/~hxu/courses/cis481/references/WhitePaperJADEEXP.pdf
2. BenHassine A, Ho TB (2007) An agent-based approach to solve dynamic meeting scheduling problems with preferences. Eng Appl Artif Intell 20(6): 857–873. https://doi.org/10.1016/j.engappai.2006.10.004
3. Chickering DM (1996) Learning Bayesian networks is NP-complete. In: Fisher D, Lenz H (eds) Learning from data: artificial intelligence and statistics V. Springer, p 121–130. https://doi.org/10.1007/978-1-4612-2404-4_12
4. Crawford E, Veloso M (2005) Learning dynamic preferences in multi-agent meeting scheduling. In: IEEE/WIC/ACM international conference on intelligent agent technology. https://doi.org/10.1109%2Fiat.2005.94
5. Doodle scheduling service website (2016). http://www.doodle.com. Accessed 14 Sep 2016
6. Garrido-Luna L, Sycara K (1996) Towards a totally distributed meeting scheduling system. In: Annual conference on artificial intelligence. Springer, Berlin. https://doi.org/10.1007/3-540-61708-6_50
7. Hossain SMM, Shakshuki EM (2013) A deliberative agent for meeting scheduling. In: 2013 IEEE 27th international conference on advanced information networking and applications (AINA). IEEE. https://doi.org/10.1109/AINA.2013.87
8. Mooney EL, Rardin RL, Parmenter WJ (1996) Large-scale classroom scheduling. IIE Trans 28(5): 369–378. https://doi.org/10.1080/07408179608966284
9. Murphy KP (2006) Naive bayes classifiers, vol 18. University of British Columbia, p 60. https://datajobsboard.com/wp-content/uploads/2017/01/Naive-Bayes-Kevin-Murphy.pdf
10. Sen S, Durfee EH (1991) A formal study of distributed meeting scheduling: preliminary results. In: ACM conference on organizational computing systems. https://doi.org/10.1023/A:1008639617029
11. Sen S, Durfee EH (1995) Unsupervised surrogate agents and search bias change in flexible distributed scheduling. In: First international conference on multi-agents systems. https://www.aaai.org/Papers/ICMAS/1995/ICMAS95-045.pdf
12. Zhang H, Jiang L, Su J (2005) Hidden naive bayes. In: AAAI. http://dl.acm.org/citation.cfm?id=1619410.1619480
13. Zunino A, Campo M (2009) Chronos: a multi-agent system for distributed automatic meeting scheduling. Expert Syst Appl 36(3): 7011–7018. https://doi.org/10.1016/j.eswa.2008.08.024