2017 IEEE 10th International Workshop on Computational Intelligence and Applications
November 11-12, 2017, Hiroshima, Japan

On the Distributed Optimization of Calendar Events

Steffan Boodhoo and Patrick Hosein
Department of Computer Science
The University of the West Indies, St. Augustine
Trinidad and Tobago
steffan@lab.tt, patrick.hosein@sta.uwi.edu

***Abstract*—Many organizations use scheduling software to assist in the organizing of events. One popular example is Microsoft Outlook. Such software typically requires event participants to vote on time slot allocations or to negotiate with one another on time slot allocations, until event attendees are appropriately satisfied.**

The process of scheduling events into time slots can be automated by requiring users to apply soft constraints on their availability for different time slots. In a soft constraint system, users assign values to time-slots, which denote the relative importance of each event. The problem can then be formalized as an optimization problem where the objective is to increase the efficiency in the scheduling of events versus availability in participants' schedules.

Since knowledge of all events is not known initially, the optimal placement of events can change over time. Therefore, there is a need for on-demand re-optimization of schedules. This can be a timely process as the scheduler has to account for the best placement of all events across all users' time slots. One method which deals with problems that require a high volume of processing is to split and distribute the problem across machines. However, since the optimal solution depends on a maximization of all events across all time-slots, an event may need to be 'aware' of other events' placements, as this can affect which slot is best. This paper presents an efficient way of distributing users, and by extension, events to distribute work while keeping relevant events together.

***Index Terms*—Event Scheduling, Electronic Calendar, Optimization**

I. Introduction

As the number of members in an organization grows, so too can the complexity of scheduling meetings and events to maximally accommodate target attendees. Consequently, such growth necessitates the development and deployment of automated mechanisms for scheduling meetings to cover the preferences over the targeted attendees.

Traditional scheduling systems are often inflexible and fail to consider the vagaries of everyday life. Many traditional systems operate in a framework of hard constraints. Hard constraint systems use binary values to denote availability. This means for any time slot, a user is either free or not. These systems are easily encumbered in circumstances where there exists no time-slot that is universally free among targeted attendees. In such a system, satisfying all targeted attendees can be impossible, thereby forcing them to sacrifice either the event being planned or their other engagements. Furthermore, as circumstances change, so too can the targeted attendees' availability for an event. Ideally, a scheduling system should be able to readily adapt the changes in the availability of targeted attendees with minimal interference.

In our previous work [1], we circumvented these problems using a system that utilized soft constraints. In a soft constraint system, users can select from a range of values (e.g. 1 to 10) to to indicate a time-slot's relative importance as opposed to marking time-slots as free or not. For instance, in a hard constraint system, a doctor's appointment or a haircut would both be placed in time-slots, which would then be marked as busy. However, in a soft constraint system, the user could assign relative degrees of importance to both events. This allows them to indicate that their haircut is much less important than their doctor's appointment and that they are more willing to have the former overwritten for an important meeting.

There are two types of events. Public, which involves more than one person (e.g. a meeting between team members), and private, which only involves the single user (e.g. a haircut). Both public and private events can be canceled and created anytime. The need for a non-traditional automated approach becomes increasingly apparent when considering the volatility of events. The value of these events when created or canceled can have a significant impact, which can reduce the efficiency of the current scheduling. In traditional approaches, once an event is scheduled, its placement never changes. However, given the volatility of events, schedules should be adapted to suit.

In reality, events of greater importance may arise, which can take the place of current events, which in turn can affect the schedules of other participants. For example, consider a meeting involving several persons. If a key participant has some new event of greater importance scheduled at the same time as the aforementioned meeting, he would likely miss the meeting. The meeting may either be barren or abandoned, leaving the other participants with a wasted/free slot that could have been used for something else. This leads to the conclusion that the optimal scheduling solution is volatile and can change with every new event added or old event removed. This problem was addressed in a prior contribution which was the ability to re-optimize schedules on demand.

Re-optimizing all schedules for large organizations can take a considerable amount of time, as for every meeting/event, for all its participants, each time-slot must be considered. Not only does an event have to evaluate the benefit (importance value) gained from its participants' time-slots, but it must also

978-1-5386-0469-4/17/$31.00

have an 'awareness' of other events as well. The proposed solution is distributed in nature, whereby the problem is split into parts, with each part being solved independently (on possibly different machines). The individual solutions are then recombined into one while addressing any complications between distributions. A simple fracturing of events may not yield an optimal solution as some events affect each other's placement. To determine the degree in which one event affects another, the participants common to both events are used. Using event audiences as the basis for interaction, the problem is then decomposed into groups of persons who share similar events. After which, each group of persons are independently solved.

II. Related Work and Contributions

This paper is an extension of the work in [1] in which we presented a system of using soft constraints, as well as a heuristic method to schedule events and re-optimize. However, this method was still very computationally intensive, which is what this paper addresses. Also, as stated in [1], there are no similar models to compare so we will revisit those most relevant.

The patent [2] presents a method in which events are scheduled based on the optimization of variables. The variables represent real world factors such as Location, Equipment, and Attendants. On any given day and time-slot, a variable is given a binary value (available or not available) to represent its availability. Variables can be given a set of constraints that can vary. For example, locations having a preference and attendants being mandatory, these variables can be categorized as preferred and optional, respectively. The scheduler tries to find the day and time such that the availability value is maximized across variables.

The patent [3] attempts to solve the problem of having the user continually manage their schedule. It attempts to do this in several ways, one of which is a combination of variables. One such variable is similar to what was presented in [1], by the use of priority level or importance score. However, while [1] tries to optimize across all persons, [3] focuses on a particular users schedule.

The patent [4] gives the user the ability to insert flexible events. A user can specify a flexible event by giving a duration for the event (how long it lasts) and a time period in which the event can be moved around. The user can also specify a minimum blocking time. This is the minimum duration in which the flexible event can be split. This is used when there are no contiguous time-slots available within the period that fits the flexible events duration.

In the patent [5], they optimize event scheduling by trying to learn/capture the users preferences. Their work can be described in three phases, a proper model to capture specific user preferences, a proper learning technique, and a reasoning/ranking system which can select the best schedule generated.

[6] is a mainstream solution to event scheduling using the traditional approach of hard constraints. The patent [7] is also traditional, however, given the similar constraints (dates, attendees, and times), it presents a list of candidate times. In the case that no candidate times can be generated, it allows for the relaxation of constraints.

The patents [8] and [9] are similar to [6]. However, [8] introduces a best fit approach which can suggest another time given extremely low attendance or find another slot for users with a higher attendance priority. [9] reutilizes the concept of best fit, however it expands its framework from users to resources meaning resource availability can be accounted for in much the same way as attendees.

Finally, patent [10] uses a traditional system. However it accommodates re-scheduling with the use of actions. The actions include shortening of meetings or finding another time-slot. Users give penalty values for actions, the scheduler then attempts to find a solution while minimizing the overall penalty score.

III. Problem Description

In this section we present the mathematical formulation of the global optimization problem. We define the following:

N = the number of participants, indexed by n,
T = the total number of time slots considered, indexed by t,
K = the total number of public events, indexed by k,
$\mathcal{E}_k$ = the set of participants to be scheduled in event k,
v_{nt} = value of a personal event scheduled by user n in slot t,
V_k = value of event k in any of its scheduled time slots,
$\mathcal{R}_k$ = range of possible starting slots for event k,
d_k = the duration of event k in slots,
s_k = first slot assigned to event k (the decision variable).
$\vec{s}$ = $\{s_1, s_2, \ldots, s_K\}$

There are two types of events, personal and public. A personal event involves one person, while a public event involves many. Since personal events may have limitations external to the calendar, the creator sets the time-slot by assigning a value v_{nt} to it, showing its relative importance (this value can be changed by the user). A public event k, has a set of participants $\mathcal{E}_k$, and the value V_k, which represents its importance to any user n in any given time-slot, given that $n \in \mathcal{E}_k$ (person n was invited to event k). A user determines their attendance of an event by trying to maximize their overall importance value, i.e., for each time-slot, a user selects the event (personal or public) which has the greatest value.

Events can last for more than one time-slot. As such, s_k, which is the first slot of the event, is used to represent the entire event. s_k, then, is the decision variable in deciding event attendance. This means, when deciding whether to attend an event k, we compare the value in the first slot (the value of s_k) with the value of V_k given that event k is scheduled to start at s_k.

Overlapping of events is allowed, which means persons may partially attend several events. In the case that overlap is undesirable, persons can use real world context to decide which of the overlapping events is most important to them to choose one in lieu of the others.

The problem can be formulated as follows:

$$\max_{\vec{s}} F \equiv \sum_{n=1}^{N} \sum_{t=1}^{T} \max \left\{ v_{nt}, \max_{\{k | n \in \mathcal{E}_k,\ s_k \leq t \leq s_k + d_k\}} V_k \right\} \quad (1)$$

$$\text{s.t.} \qquad s_k \in \mathcal{R}_k \qquad \forall k$$

The function $F(\vec{s})$ represents the total value of all users over all time-slots. For each time-slot, the user chooses the event with the greatest value. For a given user n and time-slot t, there are two scenarios; the user does not belong to any event scheduled at t, or there are one or more public events k scheduled at t i.e. $s_k \leq t \leq s_k + d_k$. In the first scenario, there are no public events, so we use the personal event value (if any) for that slot, denoted by v_{nt}. In the second, there can be several public events scheduled at t, in which case, we select the event with the greatest value ($\max V_k$), given that those events are scheduled at t, and user n was invited to the event ($n \in \mathcal{E}_k,\ s_k \leq t \leq s_k + d_k$).

Traditional calendars use binary values to represent availability. That is, a time-slot can either be available or not available. We consider these hard constraints as a slot marked as not available and cannot be used. This means, to schedule a public event, there needs to be a common available time-slot across all participants. The possibility of this decreases when considering factors such as event duration, number of participants and activity of participants. Event duration or d_k decreases the chances as one would have to find a set of contiguous time-slots of length d_k across all participants. When considering the number of participants, as the number grows there is more variability and less common available time-slots (even more so for contiguous ones). The same can be said with the degree of availability. If someones schedule is very busy, or rather has low availability, it may become difficult to schedule events as their available time-slots are very limited.

To address the varying levels of importance across events, the concept of soft constraints was introduced. This was done by giving every event (public or personal) a value which denotes its importance. This value ranges from 1 to 10 and represents the event's relative importance to a participant. The binary values in traditional calendars (available or busy) could not distinguish an important event from an insignificant one, but with the use of soft constraints, the range captures this dissimilarity. Consider two personal events, events A and event B, suppose event A was a doctor's appointment and event B was going to the gym. When using hard constraints, these two events are indistinguishable. However, by assigning a value V_n, we can tell which is more important. The distinction is important because missing one of these may have external consequences, such as prolonged sickness or a lost appointment fee (in the case of doctor's appointment) while the other can be easily rescheduled.

Given any user n and any time-slot t, there are two possibilities, either no events are scheduled, or several events are scheduled for the time-slot. If there is no event scheduled for t, user n can still assign a value v_n to show how important it is to have that free time. If there is a personal event scheduled at t, the user determines the importance of that personal event. For example, a medical appointment may be reason for a high value (closer to 10) while meeting with someone may have a lower value. If there are several public events scheduled for t, the event with the maximum value is compared with the value of the personal event (if any) and the maximum is chosen. The value of a public event is decided by the meeting organizer but this can change according to the context in a later approach. For this paper, an event can only replace another when its value exceeds the other.

Using soft constraints, given a set of users, a set of personal events (already scheduled by user), and a set of public events, the problem then lies in scheduling these public events. A public event is scheduled with the intent of obtaining the maximum value. It does this by trying to have most of its participants attend by choosing the least important slot common to its participants. Public events have a weight and duration, which means that if an event is scheduled in a spot which already contains high valued events across users, there will be little to no value gained as users would not attend. So given an event k, its weight V_k and its duration d_k, the objective is to find a set of consecutive time-slots (length d_k) which when assigned to k gives the most value gained.

The objective is to maximize the total value of users over all time-slots. As events are added, the overall value should increase (given that users can attend) or there would be no change; the total value should never go down. The way in which we select slots for events however, can significantly affect the value gained. One ordering of events may give a better solution (greater value) than another.

Consider two users with schedules consisting of only two time-slots. They have set their personal events such that their schedules are [0, 2] and [1, 9]. Each array represents users' time-slots and the values represent slot or event importance. Consider two public events A and B, whose length each span one time-slot and values are 5 and 10 respectively. In scheduling, we take the slot which gives us the maximum benefit (the largest increase in value). If we were to schedule the event A first, the user schedules would look as such: [5, 2] and [5, 9], since placing A in slot 1 adds a value of 9 vs slot 2 which only adds 3. Then, if we were to place event B, it would also be placed in slot 1, giving [10, 2] and [10, 9], overwriting A, since slot 1 adds 10 (5+5) whereas slot 2 only gives 9 (8+1). This placement yields an increase in value of 19. However, if we had placed event B first then event A, we would have ended up with [10, 5] and [10, 9], which would have yielded an increase of 22.

Hence, the order in which events were scheduled affects the increase in value. It is better therefore, to schedule all events simultaneously, rather than on a first come first serve basis. However, the determination of the optimal solution can be too computationally intensive for larger problems. In the next section we provide the centralized solution used to compromise between computation and scheduling optimization, and the results it yielded.

IV. Centralized Solution

In the centralized solution, when scheduling, all users and their events are evaluated together. To begin, we describe the traditional solution (TS), here, each event is evaluated in a linear manner. The TS operates as follows, for each event k, for all possible starting time-slots $t : t \in \mathcal{R}_k$, the benefit of placing event k in slot t is evaluated (note $\mathcal{R}_k$ is any time-slot that is not d_k slots away from the end of the day). To evaluate event k in slot t, we calculate the sum of differences, i.e. for each participant $n : n \in \mathcal{E}_k$, we sum $V_k - v_{nt}$ if $V_k > v_{nt}$ (we only add the difference when the events value is greater than that of the users time-slot).The event is then placed in the time-slot that yielded the largest sum of differences.

As described in the previous section, when scheduling, the given ordering of events (e.g. chronological), may yield a suboptimal solution. As such, the method Coordinate Ascent (CA) was used to try to mitigate potential value loss. CA can be described as follows. TS is first used to find initial placements for all events, then one at a time, each event is re-optimized, but keeping all other events fixed while doing so. The re-optimization logic stays the same as with TS. However, since all other events are already scheduled, the event that is being re-optimized is 'aware' of the other events, as their values will be used in the sum of differences. An event is considered as one degree of freedom (a coordinate) therefore CA consists of optimizing along one coordinate direction at a time hence the name.

In our previous work [1], CS was compared with TS and a hard constrained method (HS), a baseline problem was setup with the following parameters:

N = 100,
T = 18 hours x 5 days = 90,
K = 50,
$\mathcal{E}_k$ = randomly pick $p \in [0, 1]$, choose user with probability p
V_k = randomly pick from 5 to 10
v_{nt} = generated using the Markov Model
d_k = chosen randomly from 1 to 9 (maximum half day)
$\mathcal{R}_k$ = from slot 1 to slot 18-d_k each day of the week

The experiment was performed 100 times, and for each run, the following were calculated. The total value of all slots across all users, the fraction of users that attended events and the coefficient of variation. The coefficient of variance is a measure on the distribution of importance value amongst users (fairness), it is the ratio of the Standard Deviation and the Mean. Each statistic was collected over all 100 runs, averaged and normalized.

As seen, CA outperformed TS and HS in both value and attendance and was also the most fair, shown by the CoV. Parameters such as the number of events were varied to simulate increased activity in a limited number of time-slots (the degree of activity raised). This resulted in a greater margin of performance for CA which meant it scaled better. These results can be observed in [1]. While CA was successful in value and even found to be the fairest method, it's computation time presented a drawback. This problem and its solution shall be explored in the next section.

TABLE I
Performance Results for the Baseline Problem

Algorithm	Norm Value	Norm Attendance	Average CoV
HS	1.0	1.0	0.12
TS	1.4	3.8	0.08
CA	1.8	5.2	0.03

V. Distributed Solution

To cope with the computational burden, we try to distribute the work. To place a single event k, we consider all possible time-slots and for each time-slot we find the sum of differences for all of k's participants therefore $\mathcal{R}_k * \mathcal{E}_k$. For simplicity we assume d_k to be 1 which would then allow us to write $T * \mathcal{E}_k$. TS, which considers all events once, can be stated as $\sum_{n=1}^{K} T * E_k$. Finally, CA which uses TS as a base for the initial placements performs a re-optimization step, one event at a time. This can be repeated for several iterations y, and can be stated as $(y + 1) \sum_{n=1}^{K} T * \mathcal{E}_k$.

Given that the y^{th} iteration depends on the placements of the $y - 1^{th}$ iteration, CA performed simultaneously with the same data of $y - 1^{th}$ iteration will yield the same output for the y^{th}. Trying to split T across different m nodes changes the problem space. This would not only would result in m different placements for each event k which must now be evaluated by $m - 1$ comparisons, But, since there would be many events in a smaller space (T/m) it would result in many events overwriting each other in each of the m spaces. Concisely, large event domination occurs, which would lead to erroneous placements as the smaller events would add no value. To avoid this we must then split the events across several nodes rather than the time slots.

A simple fracturing of events without consideration of how they affect each other would hamper the effectiveness of CA as some events would no longer be constrained or 'aware' of others that significantly affect them. An event only affects another event if there is an intersection between the sets of persons invited. Therefore in order to split the events without removing the effectiveness of CA, we must do so while considering their audiences. In the model we chose, a person tends to have more events than an event has persons, this is so because when generating data, the number of persons invited to an event is generated within a range while the persons selected are done so at random. This means while an event has an upper limit, as the number of events increase, the average number of events a person attends increases as well. Trying to group events by the persons invited then, would lead to loosely grouped events. Therefore in order to group events, we cluster persons according to their events. We define the following variables:

M = the number of clusters, indexed by m

$\mathcal{F}_m$ = set of distinct elements in a cluster, indexed by m,
$\mathcal{C}_m$ = set of candidate cluster placements, indexed by m

The entire process consists of three main steps, group users, schedule clusters to produce candidate placements, and the evaluation and selection of candidate placements. To group users, they are treated each as a vector or set of events. As an example consider two users A and B, and the set of events 1,2,3,4, now suppose A was invited to events 1,3 and 4, and B to events 2,3 and 4. A would be represented as [1,3,4] and B as [2,3,4]. By viewing a user as a set of events, we can define their distance or how closely related one user is to another. To do this we use something called the *Jaccard Index*, which is simply the intersection over the union $A \cap B / A \cup B$. Using this as our distance function, we cluster users.

After the first step we end up with the set of M clusters, each containing a subset of users each represented by a set of events. A distinct set of all events is then retrieved from all the users in the cluster $\mathcal{F}_m$ and is scheduled using CA. This is done simultaneously across all clusters, which produces M sets of candidate placements $\mathcal{C}_m$, one for each set of $\mathcal{F}_m$ events. It is important to note that given two sets of clustered events $\mathcal{F}_i$ and $\mathcal{F}_j$ such that $i, j \in M$, that $\mathcal{F}_i \cap \mathcal{F}_j$ may not be empty as a user from cluster i may belong to the same event as one in cluster j. This means that the sets of candidate solutions $\mathcal{C}_i$ and $\mathcal{C}_j$ produced from $\mathcal{F}_i$ and $\mathcal{F}_j$ are not mutually exclusive.

Finally, given the sets of candidate placements $\mathcal{C}_m$ such that $m \in M$, for each event we evaluate their placements. For each event k, we go through each $\mathcal{C}_m$ candidate sets, when we find the set that contains a placement for k, we apply it. However if k belongs to more than one candidate set, we choose the placement which yields the maximum benefit. This is done by calculating the sum of differences for each candidate placement of k (if more than 1).

VI. Simulation

A small organization was modeled with a time frame of 20 days. We assumed a working day was 9 hours and the minimum time-slot to be 30 minutes. Since it is a working day, each time-slot is given a base importance value of 4 to represent the average importance of an individual's working time-slot i.e., how important it is to work for any given half hour. However the person may have some personal matter to attend to, this is accounted for with a probability of 0.1 for a higher value. The organization has 4 departments each containing 50 people (200 people). Since we chose to model departments, we generated two types of events, those that select persons from within a department only and those that select from the entire organization.

N = 200,
T = 18 slots x 20 days = 360,
K = 900 36900
V_k = randomly pick from 5 to 10
v_{nt} = base value of 4 with 0.1 probability of a higher value

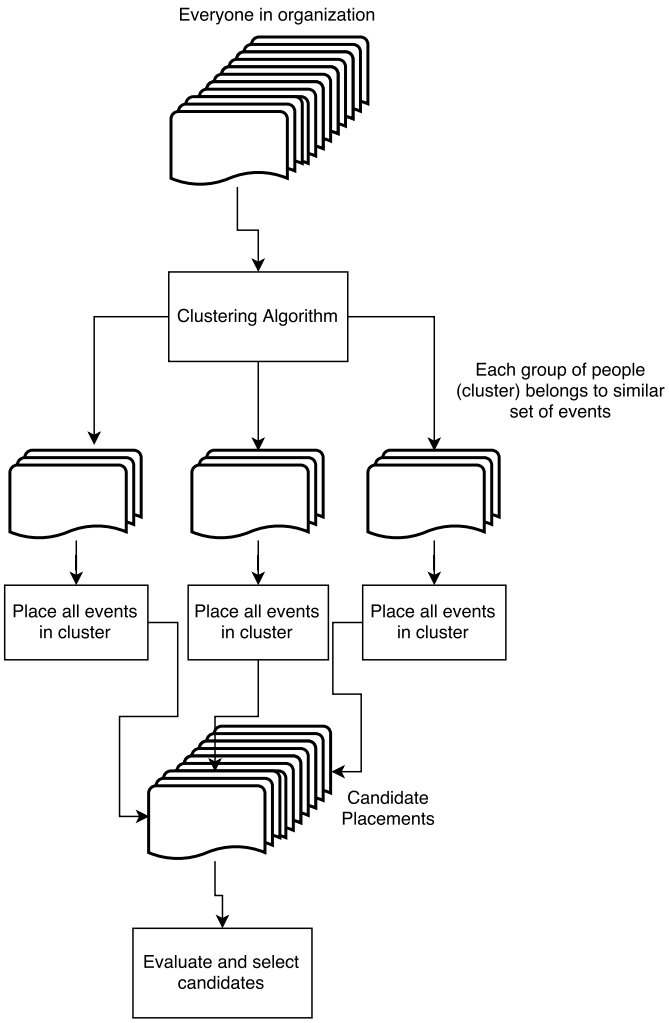

Fig. 1. Generating candidates for all elements in a cluster

d_k = chosen randomly from 1 to 9 (maximum half day)
$\mathcal{R}_k$ = from slot 1 to slot 18-d_k each day of the week

For each experiment, the initial set of data was generated to fit the description and parameters shown above. All events were scheduled using the centralized method. The total value of all slots across all users as well as the time it took to schedule events were recorded as '1 cluster'. Subsequently, the distributed method was ran with the amount of clusters as 2, the total value and time it took to schedule events were recorded as '2 clusters'. In the distributed approach, the scheduling time is calculated by adding the longest time scheduling any cluster, plus the time it took to combine the solutions between clusters after scheduling. The distributed approach is then repeated but with the clusters changing from 2 to 4 to 8 then 16. After all the cluster number variations have been calculated, we add a set number of events, then, restart scheduling from 1 to 16 clusters as previously mentioned. The experiment was repeated 100 times, the scheduling times and total values were collected and averaged for each cluster

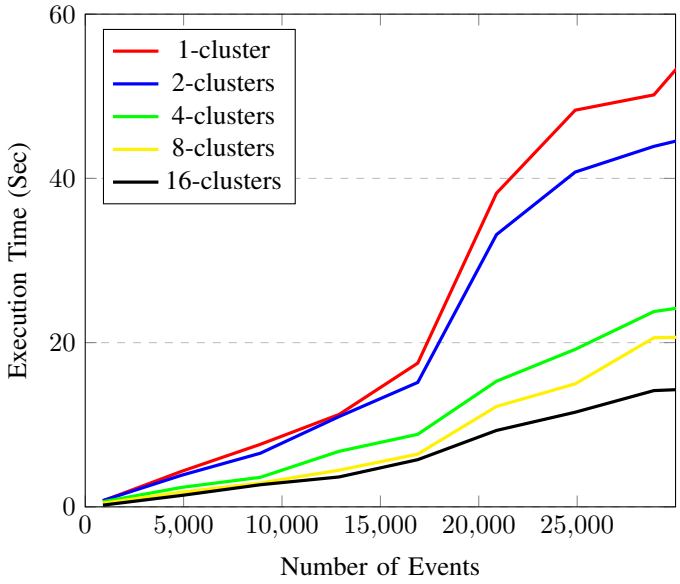

Fig. 2. Execution Time versus Problem Size for various Cluster Sizes

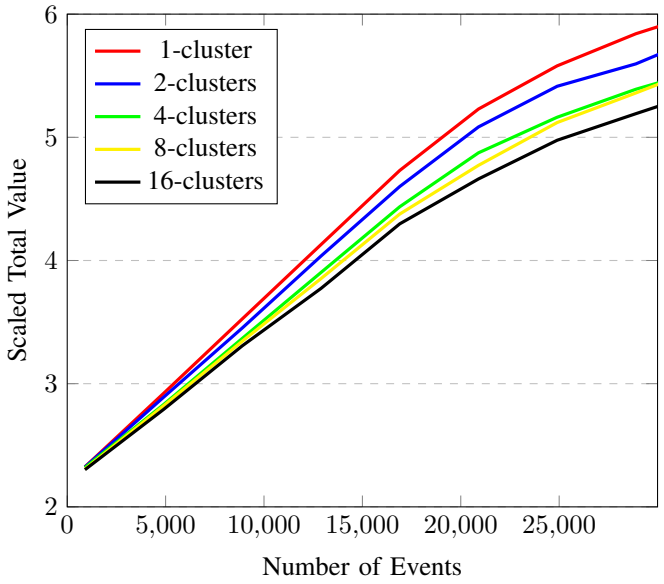

Fig. 3. Total Value versus Problem Size for various Cluster Sizes

variation for every set of K events.

Figure 2 shows the run time dependence as the number of events grows for different amounts of clusters. When varying events between the clustered and non-clustered approach there is significant improvement in time which scales with the number of clusters implemented. However, depending on the problem size, after the use of some number of clusters, the improvement does not increase significantly (if at all).

Figure 3 shows the change in total value as the number of events grows for different amounts of clusters. When splitting up the problem into clusters, some events may lose its 'awareness' of others as discussed prior. This can lead to a sub-optimal solution when cluster solutions are recombined, i.e., a decrease in value. In the given experiment, having 4 clusters seemed to provide a suitable trade-off between optimality and run time as it showed large reductions in time while being very close in the total value of the centralized method. When considering both graphs, we find that a good compromise has been found between scheduling time and scheduling optimality. The scheduling optimality refers to how close the distributed scheduling scheme is to the computed centralized schedule value (although note that the computed centralized value is also sub-optimal). When considering run time, the distributed method outperforms the centralized method by a considerable margin.

VII. Conclusions and Future Work

In our previous paper [1] we investigated a new approach to scheduling calendar events for users in which values were assigned to events. However, the approach required significant processing time and hence may not have been feasible in practice. In this paper we investigated a distributed solution to the problem that provides significant run-time improvements with relatively small reduction in optimality. The distributed approach is scalable since the number of clusters can be increased as the problem size increases so that the size of a cluster remains relatively constant. Future work includes an implementation of the proposed approach with real users and investigation of improved distributed solutions for these more realistic scenarios.

References

[1] P. Hosein and S. Boodhoo, "Event scheduling with soft constraints and on-demand re-optimization," in *2016 IEEE International Conference on Knowledge Engineering and Applications (ICKEA)*, Sept 2016, pp. 62–66.

[2] P. Capek, W. Grey, P. Moskowitz, C. Pickover, and D. Shi, "Event scheduling with optimization," Mar. 11 2008, uS Patent 7,343,312. [Online]. Available: https://www.google.com/patents/US7343312

[3] R. Zhang, O. Brdiczka, V. Bellotti, and J. Shen, "Method and apparatus for automatically managing user activities using contextual information," Mar. 6 2014, uS Patent App. 13/599,750. [Online]. Available: https://www.google.com/patents/US20140067455

[4] J. Alford, P. Arellanes, J. George, and M. Molander, "Managing flexible events within an electronic calendar," Apr. 20 2010, uS Patent 7,703,048. [Online]. Available: https://www.google.com/patents/US7703048

[5] P. M. Berry, M. Gervasio, B. Peintner, and N. Yorke-Smith, "Ptime: Personalized assistance for calendaring," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 4, p. 40, 2011.

[6] Microsoft, "Using the calendar in outlook.com," http://windows.microsoft.com/en-us/windows/outlook/calendar-use-calendar, (Accessed on 04/07/2016).

[7] J. Vincent, "Meeting scheduler with alternative listing," Sep. 17 1991, uS Patent 5,050,077. [Online]. Available: https://www.google.com/patents/US5050077

[8] D. Conmy and J. Banks-Binici, "Electronic calendar with group scheduling and automated scheduling techniques for coordinating conflicting schedules," Aug. 8 2000, uS Patent 6,101,480. [Online]. Available: https://www.google.com/patents/US6101480

[9] D. Conmy, S. Beckhardt, J. Banks-Binici, and R. Slapikoff, "Electronic calendar with group scheduling and storage of user and resource profiles," Jul. 25 2006, uS Patent 7,082,402. [Online]. Available: https://www.google.com/patents/US7082402

[10] B. Cragun and P. Day, "Method and meeting scheduler for automated meeting insertion and rescheduling for busy calendars," Oct. 16 2007, uS Patent 7,283,970. [Online]. Available: https://www.google.com/patents/US7283970