Temporal Planning for Co-Design of Host Scheduling and Workflow Allocation in Mobile Environments*

Qiang Lu^1, Yixin Chen^2, Mart Haitjema^2, Catalin Roman^2, Christopher Gill^2, Guoliang Chen^1
^{1}School of Computer Sci. & Tech., Univ. of Sci. & Tech. of China, Hefei, Anhui, 230026, China
^{2}Department of Computer Sci. & Eng., Washington Univ. in St. Louis, St. Louis, MO, 63130, USA
rczx@mail.ustc.edu.cn, {chen, mah5, roman, cdgill}@cse.wustl.edu, glchen@ustc.edu.cn

Abstract

Workflows have been successfully applied to model collaborations with a well-defined structure, which has a common restriction that the network settings are stable. Recently, a new challenging domain, collaborations among groups of hosts in an ad hoc mobile network, attracts many interests. Some key features of this application domain, such as ad hoc interactions among hosts and high levels of mobility, introduce many challenges for designing a workflow management system. The main difficulty is to design an efficient planning algorithm that automatically schedules not only workflow allocation but also the actions (movement and communication) of hosts. Existing works only consider workflow allocation but do not specify how the hosts should act to achieve the plan in a mobile environment.

In this paper, we propose a framework that co-designs the host schedule and workflow allocation in a unified way. We transform the collaboration problem into a temporal planning model and then solve it using automated planners that can also minimize the total makespan of the plan in an anytime fashion. We integrate this framework into a workflow management system CiAN. The experimental results show that our approach significantly broadens the scope of previous works by removing the requirements of knowing host schedule. Our approach is also very efficient in finding high-quality solution plans with short makespans.

Introduction

Workflows have been successfully applied to model collaborations that have a well-defined structure, which has a common restriction that the network settings are stable. (Sen et al. 2007) presents an initial investigation into the possibility of using workflows in a challenging new domain - that of supporting arbitrary collaborations among groups of hosts in a mobile ad hoc network (MANET). This application domain shares several key features: ad hoc interactions among people, high levels of mobility, the need to respond to unexpected developments, the use of locally available resources, prescribed rules of operation, and specialized knowhow. Efforts toward using workflow in ad hoc wireless environments are relatively new. Workow allocation in MANETs has wide applications such as geological monitoring, emergence coordination, and robot communities.

Designing a workflow management system (WfMS) for this domain faces many challenges, as hosts may move, and service availability may depend upon which hosts are within communication range. These challenges make most existing workflow management algorithms inadequate since they do not consider the mobility and communication constraints in ad hoc settings.

A few research efforts toward this new domain have been carried out. (Sen et al. 2007) introduces a simple heuristic allocation algorithm which gives the tasks that are harder to satisfy higher priorities to be allocated and divides the allocation of the workflows into sub-problems recursively. A workflow management system for MANETs, CiAN, is introduced in (Sen, Roman, and Gill 2008) based on this allocation algorithm. (Haitjema et al. 2010) solves this workflow allocation problem by transforming it to a numeric temporal planning problem and calling SGPlan (Chen, Wah, and Hsu 2006) to find a feasible allocation.

All the above works have a major limitation. They all assume knowing the activity schedule of each host before using an allocation algorithm to assign each task to a suitable host. However, such an assumption is very restrictive since they fix the host schedule during allocation and hence limit the decision space of the workflow allocation algorithms. The scheduling of hosts has significant impacts to the feasibility and quality of workflow allocation, since it is a main source of freedom for coping with the communication, dependency, and geometrical constraints. In an ad hoc mobile environment, two hosts usually have a communication range (e.g. the bluetooth range), and they cannot exchange data unless they are within the communication range. Since the tasks have dependencies that require notification of completion and exchange of results, such communication limits greatly complicate the workflow allocation and make the problem much harder. Existing works separate the host scheduling problem from workflow allocation. They only solve the allocation problem and do not address how to precompute a feasible host schedule. In fact, we observe that it is necessary to consider the host schedule and workflow allocation simultaneously in a **co-design** approach, in order to design a complete algorithm that can always find a feasible solution if there exists one.

*This work is supported by China Scholarship Council, NSF grants IIS-0713109, CNS 1017701, and Microsoft Research New Faculty Fellowship.

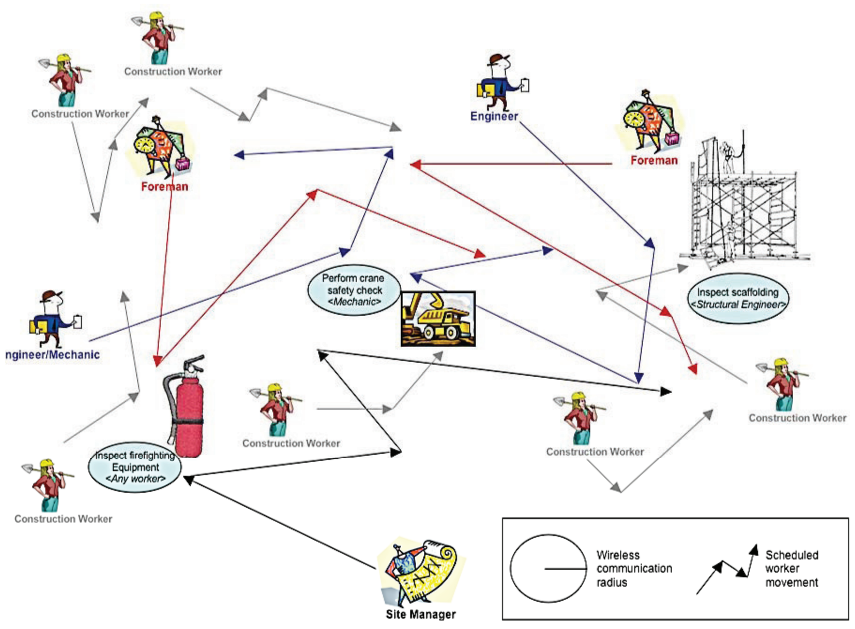

Construction Worker
Construction Worker
Foreman
Engineer
Foreman
Perform crane safety check <Mechanic>
Inspect scaffolding <Structural Engineer>
ngineer/Mechanic
Construction Worker
Construction Worker
Construction Worker
Inspect firefighting Equipment <Any worker>
Construction Worker
Site Manager
Wireless communication radius
Scheduled worker movement

Figure 1: The construction site safety report problem.

In order to address these challenges for collaboration in MANETs, we propose an approach to co-design host scheduling and workflow allocation in a unified framework. Our approach automatically translates the collaboration-in-MANETs problem to a temporal planning problem, and calls a temporal planner to find suitable host schedule and workflow allocation for WfMS. Our model can handle dynamic initial and goal states to support online insertion of new tasks. Moreover, it provides the ability to optimize the total makespan in an anytime fashion by leveraging on the advance of AI planning research.

System Model: CiAN on MANETs

We develop our planning approach on the CiAN architecture (Sen, Roman, and Gill 2008), although in principle our approach can be generalized to other workflow systems for ad hoc mobile environments. A workflow engine provides the build time environment for process modeling (definition, design, and evolution) and the runtime environment for activating, managing, and executing workflow processes (Myers and Berry 1999). Process modeling will build a library of process templates which usually integrate information flow requirements, activity decomposition, and communication constraints. In the runtime environment, it usually contains three phases:

- Process Selection: The engine will respond to some new requests by selecting and instantiating suitable process templates from the library.
- Task Allocation: Once the processes are instantiated, the engine will assign tasks to suitable processing entities according to some predefined rules or algorithms. This task allocation can be viewed as a scheduling problem, which can also be solved by automated planners.
- Enactment Control, Execution Monitoring, and Failure Recovery: The engine will maintain all the knowledge and internal control data to identify the state of each activity, transition conditions, connections among processes, and performance metrics.

In most of the existing workflow engines, they take a centralized role in coordinating the operation of processing entities, as the hosts report to the central host during execution and wait for instructions. However, designing a workflow management system targeted to MANETs faces a new challenge: coordination among the various participants becomes more complex due to the dynamic topology of the MANET which often allows only transient connectivity among hosts. To bring workflows to this dynamic type of mobile networks, CiAN is designed as a lightweight and choregraphed engine that facilitates the workflow execution in MANETs (Sen, Roman, and Gill 2008). Specifically, it uses a publish-subscribe-like protocol that takes results from a task and delivers them to the host responsible for executing the immediately succeeding tasks without going through a central coordinating entity. Note that CiAN allows decentralized information exchange during execution, but still requires centralized workflow allocation before execution.

Task Allocation Problems in CiAN

We use an example on construction site coordination to help describe the task allocation problem in CiAN. Our planning-based approach is general for workflow allocation in MANETs and not limited to this example.

Example 1 *Consider a group of construction workers moving around a large construction site like the one shown in Figure 1. All the workers are equipped with mobile devices (e.g. PDAs and smart phones) and are moving around the*

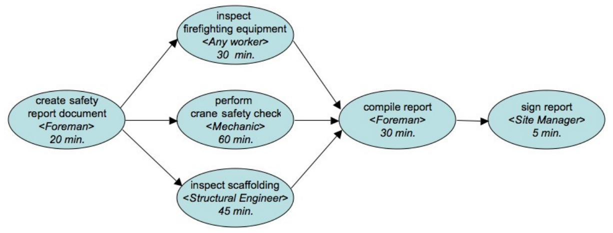


Figure 2: The workflow of the construction site safety report problem.

site working according to their individual schedules, which may be known to the management.

Now imagine that the management wants to perform a safety inspection check ad hoc and demands that a safety inspection report to be compiled. The various steps in creating the report are shown as a workflow in Figure 2.

In an ad hoc mobile environment, tasks that make up the workflow can only be done in specific locations and may require certain qualifications in order to be performed. For example, only a structural engineer working near the scaffolding can perform the inspect scaffolding task. A further complication is that after a task is executed, the results will need to be communicated to the person(s) executing the subsequent task(s) in the workflow. This transfer of results is normally done over a network. But, since the construction site is large, there may be no network infrastructure and we assume two people can only communicate when their mobile devices are within the communication range of one another. This means that even if a worker can execute a task, we must consider whether or not the worker will be able to pass the result onto the person we have chosen to perform the subsequent task. All of these constraints can make the process of choosing a worker for a given task a non-trivial problem. In fact, deciding whether or not there is an allocation of tasks to workers such that the workflow could possibly complete in this manner is NP-hard (Haitjema et al. 2010).

In the CiAN model, each task is associated with the following features:

- Locations: the locations where the task can be performed. Specified as a coordinate pair (x,y). Note that a task can be performed at one or more locations.
- Duration: the time required to execute the task.
- Hosts: the hosts who can perform the task. Note that a task can be performed by one or more hosts.
- Qualifications: a list of the qualifications necessary to perform the task. It also includes the task dependencies, a list of the tasks in the workflow that must be completed before the task to be executed. A host must know that all dependent tasks have been finished before executing the task.
- Status: whether the task has been performed.

We also have a set of workers which we will call hosts. CiAN requires to know the exact schedule of each hosts before performing task allocation, specified by the following features:

- Location: the location where the host is at for a certain time.
- Move speed: the speed of the host moving from one location to another.

The goal of the problem is to determine if there is a feasible allocation. A feasible allocation is a mapping of tasks to hosts such that each host can execute all its assigned tasks. In order for a host to execute a task t, the host must have the qualifications required by t, be in the location for the duration specified by t, and must be able to receive the results from the host(s) executing the dependencies of t. Note that the requirement that a host receives results for all the dependencies before executing its task ensures that the workflow is completed in the order specified by the workflows ordering constraints.

Limitations

CiAN requires to know the schedule of each host before performing task allocation (Sen, Roman, and Gill 2008; Haitjema et al. 2010). Each schedule entry contains a start time, location at the start time, end time, and location at the end time, which indicates when a host will be at certain location. In ad hoc mobile networks, locations of hosts may dynamically change and two hosts must meet up within a range before exchanging data when one host requires results from the other. Fixing the schedules of all hosts, CiAN can pre-compute the exact locations of moving hosts and the availability of communication between hosts for any given time. Such a strong assumption greatly simplifies the planning problem.

However, in real-world applications, a host may arrange its schedule according to the tasks it will execute. Thus, the schedules of hosts are usually unknown when performing task allocation in WfMS. Moreover, considering the host schedule and workflow allocation together gives a larger decision space which may leads to more preferable (e.g. shorter) plans. The co-design of both host schedule and workflow allocation, while considering the dependency, communication, and temporal constraints, is beyond the capability of the allocation algorithm in CiAN, or any other existing workflow algorithms we know of.

Temporal Planning for Co-Design

To overcome the limitations of the existing allocation algorithms, we formulate the host/workflow co-design problem into a temporal planning problem in PDDL and use state-of-the-art temporal planners to solve it. In our approach, we discard the assumption of knowing the host schedule *a priori* and aim at finding task allocation and host schedule at the same time.

Using the Planning Domain Definition Language (PDDL) 2.2 (Edelkamp and Hoffmann 2003), we define the temporal problem for the co-design problem as follows.

Objects. We define four objects: **host, task, location**, and **token**. Each host can execute certain tasks at certain locations. Each token is associated with a task to indicate that the task is finished. A token is consequently also an edge between two tasks representing the dependency. A host must have all the relevant tokens for a task (one for each dependency) before executing it.

Predicates. Based on the definition of objects, we define four kinds of predicates representing the status of hosts and tasks.

- **at ?h - host ?l - location**: a host h is at location l.
- **free ?h - host**: a host h is free, which means it can execute a task, move to other locations, or communicate with another host.
- **done ?t - task**: a task t is finished.
- **has-token ?h - host ?t - token**: a host h has token t, which means the host t knows that task t has been executed. Note that we use the same name for task t and the token associated with t.

Durative Actions. We consider three kinds of durative actions: an **execute** action of a host to execute a task at a given location by a given host, a **move** action of a host between two locations, and a **communicate** action between two hosts. These three kinds of durative actions are defined as follows.

- **execute ?h - host ?l - location ?t - task**: 1) Duration: a positive rational number indicating the executing time of task t. 2) Preconditions: (at start (at h l)), (at start (free h)), (over all (at h l)), and (at start (has-token h t')) for all dependent tasks t' of the task t. Since PDDL2.2 is able to describe numeric resources, we also can describe some other qualifications, such as resource requirements. 3) Effects: (at start (not (free h))), (at end (free h)), (at end (done t)), and (at end (has-token h t)). The last two effects represent that task t is finished and host h has token t.
- **move ?h - host ?l_1 ?l_2 - location**: 1) Duration: a positive rational number indicating the time for host h to move from l_1 to l_2. It equals to the distance between l_1 and l_2 divided by the speed of the host. Note that all the move actions together encode the map information of the environment. 2) Precondition: (at start (at h l_1)). 3) Effects: (at start (not at h l_1)) and (at end (at h l_2)).
- **communicate ?h_1 h_2 - host ?l_1 ?l_2 - location ?t - task**: 1) Duration: a positive rational number indicating the time for transferring a message (token t). 2) Preconditions: (at start (at h_1 l_1)), (at start (at h_2 l_2)), (at start (free h_1)), (at start (free h_2)), (at start (has-token h_1 t)), (over all (at h_1 l_1)), and (over all (at h_2 l_2)). 3) Effects: (at start (not (free h_1))), (at start (not (free h_2))), (at end (free h_1)), (at end (free h_2)), and (at end (has-token h_2 t)). The last predicate represents that the host h_2 has the token t after communicating with h_1.

The precondition (over all (at h l)) indicates that the the host h cannot move to other locations when executing a task or communicating with another host. The precondition (at start (free h)) and effect (at start (not (free h))) guarantee that the host h cannot perform two actions at the same time. Note that since dependent tasks are different in different *execute* actions, we cannot define all *execute* actions in an ungrounded way. Therefore, all actions are grounded in the domain definition of PDDL.

Initial State. The initial state of the problem includes predicates: 1) (free ?h - host) and (at ?h -host ?l - location) which indicate that host h is free now and at location l. 2) (at n (free ?h - host)) and (at ?h - host ?l - location) which indicate that host h will be free at time n (a positive rational number) and at location l then. This timed initial literals, (at n (free ?h - host)), is a feature of PDDL2.2 (Edelkamp and Hoffmann 2003) which represents facts that become true or false at certain time points.

Note that we use this feature to support dynamic planning. During the execution of a plan, when a set of new task requirements come in, a host may be executing some tasks currently and will be free in the future time n. If the host is required for a task, we need to add the host as an object in the planning problem. Thus we use the timed initial predicates to represent the initial status of such hosts. By exploiting the timed initial literals in PDDL2.2 planning, our approach can support dynamic planning in response to new tasks during execution.

Goal State. The goal state includes predicates (done ?t - task) for all required tasks t. Again, our approach can support dynamic planning. When new tasks are added during execution, we generate a new planning problem with the dynamic initial state discussed above and goal state using these new tasks, and then call a planner to solve it.

Based on the above PDDL model, any solution plan found by a temporal planner will give a co-design solution, which specifies the workflow allocation as well as a schedule of movement and communication for each host.

The resulting temporal problem is temporally simple without required concurrency (Cushing et al. 2007). A temporal problem has required concurrency (called temporally expressive) when, in any solution, there exist two actions a_1 and a_2 such that: 1) a_1 has two effects (at start f) and (at end (not f)) (which means f is an interval add-effect), and 2) a_2 has a precondition (at start f). These two conditions require a_1 and a_2 to be executed concurrently. Clearly, in our PDDL2.2 model, no two actions satisfy these two conditions. Note that temporally simple problems are typically more tractable than temporally expressive ones. Hence, our formulation can be efficiently solved using existing temporally simple planners.

Note that although the problem is temporally simple, it is essential to exploit the durative natures and concurrency of actions in our model in order to generate efficient plans with short makespans. In real-world applications, the temporal feature is very important because users usually want to finish all tasks as soon as possible. In our implementation, we use an anytime temporal planner, TFD (Eyerich, Mattmüller, and Röger 2009), to optimize the makespan.

We show three grounded actions of Example 1 in Figure 3.

The optimal solution found by Temporal Fast-Downward (TFD) (Eyerich, Mattmüller, and Röger 2009) is shown as follows. The first column is the scheduled time of each action and the last column is the action duration. The

```
(execute Foreman-0 crane-zone compile-report
  :duration (= ?duration 30)
  :condition (and
    (at start (at Foreman-0 crane-zone))
    (at start (free Foreman-0))
    (at start (has-token Foreman-0 inspect-firefighting-equipment))
    (at start (has-token Foreman-0 perform-crane-safety-check))
    (at start (has-token Foreman-0 inspect-scaffolding))
    (over all (at Foreman-0 crane-zone))
  )
  :effect (and
    (at start (not (free Foreman-0)))
    (at end (has-token Foreman-0 compile-report))
    (at end (done compile-report))
    (at end (free Foreman-0))
  )
 )


(communicate Foreman-0 crane-zone Site-Manager crane-zone compile-report
  :duration (= ?duration 1)
  :condition (and
    (at start (at Foreman-0 crane-zone))
    (at start (at Site-Manager crane-zone))
    (at start (free Foreman-0))
    (at start (free Site-Manager))
    (at start (has-token Foreman-0 compile-report))
    (over all (at Foreman-0 crane-zone))
    (over all (at Site-Manager crane-zone))
  )
  :effect (and
    (at start (not (free Foreman-0)))
    (at start (not (free Site-Manager)))
    (at end (free Foreman-0))
    (at end (free Site-Manager))
    (at end (has-token Site-Manager compile-report))
  )
 )

(move Foreman-0 crane-zone firefighting-zone
  :duration (= ?duration 2)
  :condition (at start (at Foreman-0 crane-zone))
  :effect (and
    (at start (not (at Foreman-0 crane-zone)))
    (at end (at Foreman-0 firefighting-zone))
  )
 )
```

Figure 3: Three grounded actions of Example 1.

total time (makespan) of the solution is 120.18 seconds.

– 0.01 (move site-manager init-loc crane-zone) [2]

– 0.02 (move foreman-1 init-loc crane-zone) [2]

– 0.03 (move construction-worker-4 init-loc firefighting-zone) [3]

– 0.04 (move engineer init-loc scaffloding-zone) [3]

– 0.05 (move mechanic init-loc crane-zone) [2]

– 2.06 (execute foreman-1 crane-zone create-safety-report-document) [20]

– 22.07 (communicate foreman-1 crane-zone mechanic crane-zone create-safety-report-document) [1]

– 23.08 (execute mechanic crane-zone perform-crane-safety-check) [60]

– 23.09 (communicate foreman-1 crane-zone construction-worker-4 firefighting-zone create-safety-report-document) [1]

– 24.10 (execute construction-worker-4 firefighting-zone inspect-firefighting-equipment) [30]

– 24.11 (communicate foreman-1 crane-zone engineer scaffloding-zone create-safety-report-document) [1]

– 25.12 (execute engineer scaffloding-zone inspect-scaffolding) [45]

– 54.13 (communicate construction-worker-4 firefighting-zone foreman-1 crane-zone inspect-firefighting-equipment) [1]

– 70.14 (communicate engineer scaffloding-zone foreman-1 crane-zone inspect-scaffolding) [1]

– 83.15 (communicate mechanic crane-zone foreman-1 crane-zone perform-crane-safety-check) [1]

– 84.16 (execute foreman-1 crane-zone compile-report) [30]

– 114.17 (communicate foreman-1 crane-zone site-manager crane-zone compile-report) [1]

– 115.18 (execute site-manager crane-zone sign-report) [5]

Task execution

Once the planning process is completed, the execution of the workflow begins. When an agent is assigned an action, it adds an entry to its schedule that contains all the necessary information to execute the appropriate service as directed by the manager. The agent is free to roam, but is responsible for completing assigned tasks, including moving to certain locations, executing tasks, and exchanging task execution information. The execution phase of this workflow is in a decentralized and distributed manner.

The flexibility of such a dynamic domain is much higher than normal workflow problems, due to the high variability of agent speeds and task completion probability. The change of speed may cause an agent not able to arrive at a required location in time for executing tasks or communicating with another agent. A failed execution of a task may break the dependencies of other tasks and cause the whole plan to crash. Thus, the ability to handle these execution exceptions is very important for the workflow management system.

In our system model, we can handle these exceptions since our planning algorithm, as we discussed in this paper before, can support dynamic planning by exploiting the timed initial literals. During the execution of a plan, when an agent fails to execute an action, it will first try to re-do it if there is enough time before the start time of the next action in its schedule. The replanning process will be triggered if the re-do fails. We first collect the current state of free agents and timed future state of working agents if necessary to generate the new initial state, and the unfinished goal tasks to generate the new goal state. Then we generate a new planning problem with the dynamic timed initial state and goal state, and then call a planner to solve it. As we will show in our experimental results, we can solve the planning problem efficiently (under 10 seconds for the largest problem with 30 hosts). Hence, it is feasible to plan dynamically during execution.

Related Work

AI Techniques for Workflow Management

An overview of early uses of AI techniques in workflow engines is presented in (Myers and Berry 1999). The paper overviews three major areas: 1) reactive control systems providing adaptive process management, 2) AI scheduling providing adaptive resource allocation, and 3) AI planning providing process synthesis and repair (with a focus on replanning).

There are several other work on applying AI planning techniques to WfMS. (Moreno and Kearney 2002) describes an integration of AI planning techniques with an existing workflow management system. It uses AI planning to automatically generate a sequence of instantiated activities. (Schuschel and Weske 2003) outlines a framework for an integrated planning and coordination system, which uses AI planning to support business processes. (Shi, Yang, and Sun 2011) presents a workflow management system with dynamic goal tasks, using AI planning to solve these goal driven workflow planning problems. However, none of these works considers ad hoc mobile environments and the planning of host activities as this paper does.

WfMS with Temporal Constraints

Modeling temporal constraints in WfMS is first proposed in (Marjanovic and Orlowska 1999; Eder, Panagos, and Rabinovich 1999). (Marjanovic and Orlowska 1999) proposes a framework for time modeling in production workflows. Relevant temporal constraints are presented, and rules for their verification are defined. Furthermore, to enable visualization of some temporal constraints, a concept of “duration space” is introduced. (Eder, Panagos, and Rabinovich 1999) proposes modeling primitives for expressing temporal constraints between activities and binding activity executions to certain fixed dates. It presents techniques for checking satisfiability of temporal constraints, and enforcing these constraints at run-time. The techniques for temporal constraint management are based on the *timed activity graph*.

(Son and Kim 2001) proposes a scheme to maximize the number of workflow instances satisfying given deadlines. It develops a method to determine the minimum number of servers (MNS) for any critical activities, an activity that should be finished without delay for a given input arrival rate. (De Maria, Montanari, and Zantoni 2005) proposes to use the finite timed automata as a tool to specify timed workflow schemas and to check their consistency. Temporal constraints are often set when complex e-science processes are modeled as scientific workflow specifications. (Chen and Yang 2010) systematically investigates how to localize a group of fine-grained temporal constraints so that temporal violations can be identified locally for better cost effectiveness. Most of these works are designed to derive or check certain properties of a temporally constrained system. However, they cannot be directly used to solve workflow allocation problems, nor the more complex co-design problem. An interesting future work is to incorporate these analysis as heuristic guidance and pruning conditions to further improve our planning approach.

Experimental Results

We evaluate our method by designing a simulator. In our implementation, we use two state-of-the-art temporal planners, SGPlan (Chen, Wah, and Hsu 2006) and TFD (Eyerich, Mattmüller, and Röger 2009), to solve the compiled temporal planning problem in PDDL2.2. We generate a series of random problems and measure the time taken to find a solution plan and the temporal makespan. The main parameters of generated problems include:

– n_h: the number of hosts.

– n_t: the number of tasks.

– n_c: the number of dependencies for each task.

– max_x, max_y: location range ($0 \leq x \leq max\_x$, $0 \leq y \leq max\_y$) that hosts may work at. We set $max\_x = max\_y = 400m$.

– s: the moving speed of hosts. We set it to 1.7 m/s which is close to human walking speed.

– d: the duration of actions. The duration of *execute* is randomly chosen from [60, 300] seconds, the duration of *communicate* is set to 1 second, and the duration of *move* is set to the distance of two locations divided by speed ($dis(l1, l2)/s$).

n_h, n_t, and n_c are the main parameters deciding the complexity of the generated problems. In (Sen et al. 2007), the largest problem's parameters are $n\_h = 12$, $n\_t = 2{*}n\_h$, and the probability for each task to be assigned a task dependency equals to 0.3. Since the host schedule in (Sen et al. 2007)'s testing problem is randomly generated and a large number of task dependencies may lead to a low chance of finding a feasible task allocation, it sets a low probability of having task dependencies. In our evaluation, we generate a series of problem with $n\_h \in [1, 30]$, $n\_t = 2 * n\_h$, and n_c is randomly chosen from [0,3]. The dependencies between tasks are randomly generated with the guarantee that no cycle exists. We set this higher dependency probability than (Sen et al. 2007) because real-world applications usually require a large number of task dependencies as shown in Example 1.

All experiments are conducted on a workstation with 2.8GHz CPU and 2GB memory. The running time limit for each problem instance is set to 30 seconds. We set this relatively low time limit in order to ensure the practicability of our approach in real-world applications, where users prefer short planning time.

From experiments, we see that our approach can solve the co-design problems efficiently. Specifically, it requires no more than 2 seconds to find a solution on any problems with up to 30 hosts and 60 tasks, a problem size much larger than the largest problem previous work considered (12 hosts and 24 tasks) (Sen et al. 2007). The running time and makespans are shown in Figure 4.

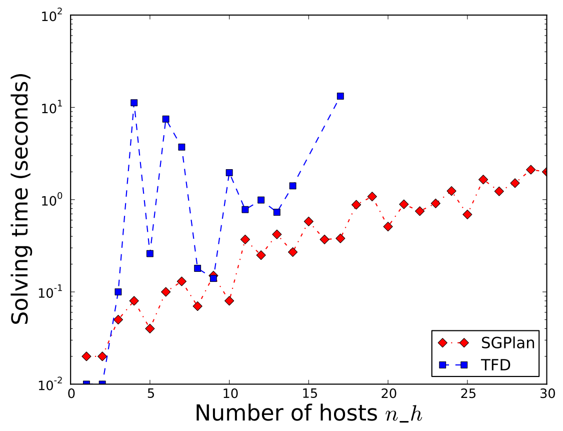
Since the allocation algorithms in (Sen et al. 2007; Haitjema et al. 2010) and our approach are based on different assumptions, we cannot compare them directly using our testing problems. However, some quantitative comparisons can be made here. Comparing our approach against the allocation algorithm in (Sen et al. 2007), we found that our approach is still very efficient even though it is solving a much harder problem (which considers host scheduling under a higher degree of task dependencies). For example, for a problem with $n\_h = 12$ and $n\_t = 24$, the mean time to perform CiAN's allocation algorithm is about 1.2 seconds (Sen et al. 2007) while the solution time of our method is 0.25 second.

(Haitjema et al. 2010) usually cannot find a feasible allocation on our problems where $n\_h \geq 8$ as the randomly generated host schedule. This clearly shows the limitation of that approach and the need for co-design. Also, it spends more time to solve allocatable problems than our approach. For a problem with $n\_h = 4$ and $n\_t = 8$, using the same planner SGPlan, it takes 3.74 seconds to find an allocation while our approach takes only 0.03 second.

Another advantage of our method is that we can minimize the temporal makespan (the total time of finishing all tasks) by using TFD, an anytime planner which can optimize the makespan. Figure 4 shows the solution time and makespan of a feasible solution found by SGPlan and of the best solution found by TFD under 30 seconds. Obviously, SGPlan is faster and solves more problems than TFD under 30 seconds. On the other side, TFD usually finds better plans (with hundreds of seconds shorter makespans) at the cost of more planning time. For example, on the problem with $n\_h = 10$, TFD finds a plan with a makespan $2313s$ in 1.96 seconds while SGPlan found a solution with a makespan $4384s$ in 0.08 second. Considering the large saving of execution time, users are likely willing to spend a little more time on planning.

Conclusions and Future Work

Workflows have been extensively studied and applied to model collaboration in well-defined and stable networks. In this paper, we consider a new challenging environment of ad hoc mobile networks for workflow allocation. Existing



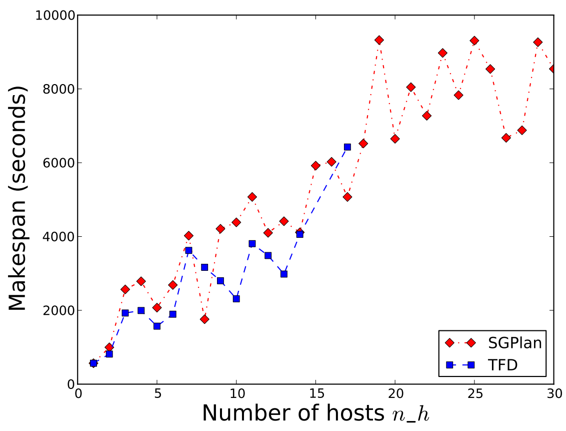


Figure 4: The running time and solution quality (in terms of makespan) of SGPlan and TFD. Note that $n\_t = 2 * n\_h$.

workflow engines for this domain can find a workflow allocation based on the restrictive assumption of knowing and fixing schedules of all hosts a priori. In this paper, we have presented a framework that co-designs the host schedule and workflow allocation in a unified way. We transform the collaboration problem into a temporal planning model and use automated planners to solve it efficiently. Our experimental results show that it is practical to use temporal planners to find feasible and even optimized schedule that coordinates hosts and workflows together under complex temporal, communication, and dependency constraints.

A practical workflow allocation algorithm on MANETs has many applications. Responding to catastrophes such as chemical spills, conducting geological surveys of remote areas, and even managing a community of robots exploring hazardous areas are only a few examples of the many activities that would potentially benefit from this approach. We plan to explore such real applications in our future work.

References

Chen, J., and Yang, Y. 2010. Localising temporal constraints in scientific workflows. *Journal of Computer and System Sciences* 76(6):464 – 474.

Chen, Y.; Wah, B. W.; and Hsu, C.-W. 2006. Temporal planning using subgoal partitioning and resolution in sgplan. *Journal of Artificial Intelligence Research* 26:323–369.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal? In *Proc. of IJCAI*.

De Maria, E.; Montanari, A.; and Zantoni, M. 2005. Checking workflow schemas with time constraints using timed automata. In *On the Move to Meaningful Internet Systems 2005: OTM Workshops*, volume 3762, 1–2.

Edelkamp, S., and Hoffmann, J. 2003. Pddl2.21 the language for the classical part of the 4th international planning competition. *Technical Report N. 194, ALbert Ludwigs University Institue for Informatik, Freiburg, Germany*.

Eder, J.; Panagos, E.; and Rabinovich, M. 1999. Time constrains in workflow systems. In *Proc. of CAiSE*, 286–300.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *ICAPS*.

Haitjema, M.; Chen, Y.; Roman, C.; and Gill, C. 2010. Automated planning: Workflow allocation in a mobile setting. *http://www.cse.wustl.edu/ chen/workflow/*.

Marjanovic, O., and Orlowska, M. E. 1999. On modeling and verification of temporal constraints in production workflows. *Knowl. Inf. Syst.* 157–192.

Moreno, M. D. R., and Kearney, P. 2002. Integrating AI planning techniques with workflow management system. *Knowledge-Based Systems* 15(5-6):285 – 291.

Myers, K. L., and Berry, P. M. 1999. At the boundary of workflow and AI. In *Proc. of AAAI*.

Schuschel, H., and Weske, M. 2003. Integrated workflow planning and coordination. In *In 14th International Conference on Database and Expert Systems Applications*, 771–781. Springer-Verlag.

Sen, R.; Hackmann, G.; Haitjema, M.; Roman, G.-C.; and Gill, C. D. 2007. Coordinating workflow allocation and execution in mobile environments. In *COORDINATION*, 249–267.

Sen, R.; Roman, G.-C.; and Gill, C. D. 2008. Cian: A workflow engine for manets. In *COORDINATION*, 280–295.

Shi, Y.; Yang, M.; and Sun, R. 2011. Goal-driven workflow generation based on AI planning. *Computer and Computing Technologies in Agriculture IV* 346:367–374.

Son, J. H., and Kim, M. H. 2001. Improving the performance of time-constrained workflow processing. *Journal of Systems and Software* 58(3):211 – 219.