

Distributed Allocation of Workflow Tasks in MANETs

Rohan Sen, Gruia-Catalin Roman, and Christopher Gill
Department of Computer Science and Engineering
Washington University in St. Louis

Campus Box 1045, One Brookings Drive, St. Louis, MO 63130, U.S.A.

Email: {rohan.sen, roman, cdgill}@wustl.edu

Abstract

While most workflows and WfMSs today are designed under the assumption that the participants are connected to each other across a stable wired network, the jump to a scenario where participants are connected to each other across more dynamic networks e.g., mobile ad hoc networks (MANETs) requires novel distribution and management schemes in order to make the WfMS work. In this paper, we describe a process for the allocation of workflow tasks in MANETs where there are no centralized entities, hosts are physically mobile, communication links are frequently interrupted, and spatiotemporal considerations become increasingly important. Our solution is a distributed process that uses a combination of heuristics and constraints to arrive at allocation decisions. An evaluation of our approach is also presented.

1 Introduction

Workflows are useful for modeling large, well-structured activities that can be broken down into smaller tasks and performed by multiple *participants*. Typically, there are three stages in the life of a workflow: (1) Specification, which is the formulation of the workflow in a machine parseable language, (2) Allocation, which is the assigning of tasks in the workflow to participants that have the skills to perform those tasks, and (3) Execution, which is the performing of those tasks by the participant that have been assigned to them and the collation of results. Popular languages for specifying workflows today are BPEL [2] and XLANG [21], among others. The process of

allocation and execution is handled by Workflow Management Systems (WfMSs) such as ActiveBPEL [1], BizTalk [3], and Groove [4], to name a few.

These systems, and indeed most mature WfMSs today share the common characteristic that their operating environment is relatively static, i.e., the network topology is stable, links are reliable, the set of participants providing services does not change rapidly over time due to the fact that they are located on large enterprise servers. However, at the level of the workflow model itself there is no restriction that prevents it from being used in more dynamic settings such as mobile ad hoc networks (MANETs), where applications modeling collaboration among workers at a construction site, coordination among a team of geological surveyors in a remote region, or coordination between teams working at the scene of a toxic spill are all possible. Rather, it is the allocation processes and current WfMSs (which by design are intended to work in stable environments) that keep workflow from being applied in more dynamic settings. There are several efforts to develop an execution engine for workflows in mobile settings such as [20, 17, 7]. In this paper, we focus on the steps needed before the workflow can be executed, i.e., the process by which tasks are allocated to participants that perform them.

Our approach consists of taking the centralized allocation process consisting of a single coordinating entity and synchronous service calls and transforming it into a distributed and asynchronous process. We describe a distributed allocation algorithm that takes a monolithic workflow and fragments it into smaller “sub-workflows” using a set of pre-defined rules. Each fragment is then assigned to a *local coordinator*, a spe-

cial participant that is responsible for allocating a related subset of tasks that are assigned to it. The allocation process is designed to work in a “just-in-time” manner, with tasks being allocated just before they need to be performed. We also discuss the tradeoffs, such as constraints on motion of participants, policies for workflow partitioning, degree of workflow distribution, etc. that need to be made for our approach to work effectively and analyze the effects of these tradeoffs through simulation experiments.

Our approach is beneficial because: (1) the decentralized approach mitigates the problem of a single point of failure, crucial when operating in dynamic MANETs in particular and (2) the just-in-time nature of the algorithm removes the requirement that all tasks be assigned *a priori*, which reduces the number of early (and potentially incorrect) decisions.

2 Related Work

In current deployments of WfMSs, such as ActiveBPEL [1], the allocation process is centralized. This is possible because these systems almost always run in a corporate LAN environment or across the Internet where wired connections guarantee reliable and instantaneous communication between the centralized WfMS and participants in the workflow. This in turn makes centralized workflow management architectures practical. Even WORKPAD [8], which is designed for mobile networks, uses a centralized WfMS.

In this environment, allocation is typically handled as follows: the WfMS selects a task to execute. It then looks up a central directory for a suitable software service to perform the task (in some cases the services may even be hardcoded, e.g., BPEL’s `partnerLinks`). Once a service has been found, the WfMS sends the input data to the service and waits for a response. When the response arrives, it selects a service for the next task and repeats the procedure.

In MANETs however, there is neither the opportunity to have a centralized management architecture, nor are the participants always accessible due to wireless links that might break frequently due to host mobility. Hence the allocation process is much more opportunistic in nature, allocating tasks *a priori*, whenever hosts are within communication range, which is a

different approach from those in use today.

In a sense, our task allocation problem is similar to the Job Shop Scheduling (JSS) problem where a set of jobs is scheduled on a set of machines such that no machine executes more than one job at a time and the total duration for executing the jobs is minimized. In our work, the tasks and participants are analogous to the jobs and machines respectively.

The difference lies in the fact that our primary objective function is to maximize allocation. Minimizing the time required to complete the jobs is only a secondary objective. Also, in addition to jobs being admitted during the scheduling process (the entire workflow is not available for scheduling up front), we also have to accommodate the possibility of additional machines being admitted during the scheduling process. Finally, our approach must also take into consideration the constraints imposed by the physical mobility of hosts and the fact that the machines are heterogeneous (all jobs cannot be scheduled on all machines).

In [6], the authors describe a heuristic-based method for solving the basic JSS problem while [10] describes a genetic approach to solving the same problem. More pertinent to our work is [16], which considers the JSS problem with availability constraints, i.e., where the set of available machines on which to schedule jobs changes over time. This is analogous to the reachability of hosts changing over time in a MANET. Another piece of work that is closely related to our effort is reactive JSS [14] where the schedule is not computed *a priori* but over a period of time. More recently, researchers have considered the use of neural nets to solve the JSS problem [22].

Another related area is robot task scheduling. In [12], the authors propose a taxonomy of multi-robot task assignment problems. Our work is closest to the extended time assignment variants of the problem proposed therein. Solutions to this problem involve using a market-based economic model [23], an auction-based approach [11] that uses concept of task utility and fitness of a robot to perform a task to make allocations. Incorporation of spatiotemporal considerations including the formation of organizations and a reward scheme is described in [5] while a scheme for fault tolerant coalition formation is described in [18]. Similar approaches have also been used to allocate resources in wireless sensor networks [15].

3 Computational Model

In our work, we assume that there exists a workflow which needs to be allocated and subsequently executed in a well-defined physical space. Each task in the workflow has an earliest start time, a deadline, a duration, a location at which it is to be performed, and a set of qualifications that a participant must have in order to perform it. A participant is a human who carries a mobile device such as a PDA on his/her person. We assume that there is a closed set of *participants*, each of which has a set of qualifications. The set of qualifications of a participant is a union of the skills of the human user and the software capabilities of the mobile device. In the remainder of this paper, we use the term participant and host interchangeably.

Hosts have a maximum velocity at which they can move, and a schedule. Entries in a host's schedule indicate its commitments. A host's schedule includes commitments related to workflow tasks or external commitments (presumably of a personal nature). Each entry in the schedule has a start time, end time, and the location of the host at each of these times. A host's schedule may not contain workflow-related commitments initially, it may contain personal commitments which must be considered when assigning tasks.

A small subset of hosts undertake the function of *coordinators*. Each coordinator is strongly associated with a specific sub-area of the space in which the workflow is being executed and is responsible for allocating a part of the overall workflow. Coordinators differ from "worker" hosts in the following ways: (1) they broadcast their schedule so that all worker hosts know where a coordinator is at all times, (2) they advertise a "home" location within the sub-area they are responsible for, which is the location at which they remain unless their schedule requires them to be elsewhere, (3) they do not execute any tasks, instead running the allocation algorithm as their permanent "task". The coordinators are chosen *a priori*.

Worker hosts in our system move subject to certain basic constraints. During time periods where there is an entry in their schedule, they are required to be at the location dictated by that schedule entry. An entry in the schedule is, in other words, a *commitment* by the host to be at a particular place at a particular time. If a task is allocated to a worker host, it blocks off the time

required to perform the task on its schedule, so that it is not assigned another conflicting task. A host is considered to be qualified to perform a task if it has all the qualifications and is available during the time that the task must be performed. Also it must have sufficient time to travel to and from the location of the task without exceeding its maximum velocity and without encroaching on any of its other appointments.

Our algorithms are designed to operate across MANETs, a special class of networks that do not rely on any external or fixed infrastructure. The network infrastructure is borne by the hosts that comprise the network. Due to the physical mobility of hosts, the network topology is dynamic, with communication links being available only intermittently. The transient nature of the links makes multi-hop routes expensive to maintain and prone to failure. Hence, the most reliable way to communicate is by having hosts directly connect to each other when within communication range of each other. This opportunistic style of communication fosters a decoupled style of computing. Decoupled computing means that hosts may not be reachable at all times. Hence, it is not possible to interact with the host *on-demand*, as is the case with current WfMSS that run in wired networks and have a reliable link to every host. This makes only a very coarse grained control possible over the host's activities and behavior.

4 Allocating Tasks to Mobile Hosts

Our approach is divided into three phases: 1) pre-processing steps, which occur prior to the actual allocation process, 2) the core allocation process that is agnostic to whether it is run in a centralized or distributed fashion, and 3) additional resources that distribute the core process and manage the mobility of participating hosts. We describe each phase in turn.

Preprocessing Steps. The first preprocessing step is to assign each task a time-dependent utility value that represents how critical it is that the task be allocated. For a task T at time t , its utility $U_T(t) = 1/(E_T - t)$ where E_T is the earliest starting time for task T , and t is the current time. Thus, the further in the future a task's start time is, the lower its utility to the progression of the workflow *at the current time*. Once we have computed the utility of tasks, the next step is to fragment the monolithic workflow so

that tasks can be allocated by multiple coordinators in a distributed fashion. There are two ways in which the workflow can be fragmented:

k-Minimum Cut. We assume that k coordinators are available to allocate tasks. We use a graph traversal algorithm that sorts tasks into buckets according to their depth in the graph. The k -minimum cut approach considers the combined size of adjacent buckets in turn. Cuts are made between the k bucket pairs that have the lowest combined value. The exception to this rule is cuts that would result in a fragment having lower than f tasks, where f is a user-defined parameter with a value less than N/k where N is the total number of tasks in the workflow. In such a case, the next higher cutting point is chosen.

Geographic Cut. Once again, we assume that k coordinators are available. The area in which the workflow is to be executed is divided into k zones. Using a graph traversal algorithm, we examine the task location and put the task specification in a bucket that corresponds to the zone containing the task's location. Thus, the tasks in any fragment are geographically related, i.e., they are in a subset of the total area.

Each approach has its benefits. The k -Minimum Cut keeps blocks of contiguous tasks under the responsibility of one coordinator, which is useful when recovering from localized errors (not covered in this paper). The geographic cut allows geographically related tasks to be handled by one coordinator. Since each coordinator is responsible for a specific sub-area, the geographic cut allows correlation between the location of the coordinator and the tasks they are allocating.

Core Allocation Process. At the start of the core allocation process, the coordinator sorts the tasks according to their utility. The sorted tasks are placed in a *task list* and each task is marked "yellow" to indicate that it needs no attention. The time parameter t to compute the utility of tasks is the start time of the workflow. The coordinator then runs the allocation process continuously until all tasks are allocated. The allocation process is split into three phases: (1) distributing solicitations to perform tasks to worker hosts and generating bids, (2) allocating a task provisionally to a host and revisiting the allocation decisions over time. Each step is described in detail below.

Distributing Solicitations and Generating Bids. For each task, the coordinator formu-

lates a solicitation which is a 6-tuple of the form $\langle \text{String:taskName}, \text{List:capabilities}, \text{Location:taskLocation}, \text{Time:duration}, \text{Time:start}, \text{Time:deadLine} \rangle$. Each of the six pieces of information in the solicitation can be obtained from the task specification [19]. When a worker host comes within communication range of the coordinator, the solicitations are sent to that worker host (the detection of the presence of a worker host is done by an external package that uses a system of beacons and acknowledgments to establish proximity). When a worker host receives the list of solicitations from the coordinator, it analyzes them to determine whether it is suited to perform any of the tasks advertised. If so, it submits bids for the tasks it can perform.

For each solicitation, the algorithm checks whether the capabilities required by the task is a subset of the host's capabilities. If so, it checks that host's schedule to ensure that the host does not have any previously scheduled commitments at the time that the task described in the solicitation needs to be performed. This is done using the AVAILABLE function on the host's schedule which returns a boolean value. If this check is successful, then the host is qualified *and* available to do the task. Finally, we factor in travel time. For this, we get the tasks that would immediately precede and succeed the task under consideration, were it to be assigned to this host. This is done using the GETPRECEDINGTASK and GETSUCCEEDINGTASK functions respectively. We then compute the velocity at which the host would need to travel from the location of the preceding task to the location of the task under consideration, and then on to the location of the succeeding task. If both these velocities are lower than the maximum velocity capability of the host, then it is eligible to submit a bid for that task.

Before the submission, it calculates the fraction of its capabilities that it will use in performing the tasks. It then creates a bid, which is a 5-tuple of the form $\langle \text{double:capabilityFraction}, \text{double:precedingVelocity}, \text{double:succeedingVelocity}, \text{double:maxVelocity}, \text{Time:deadline} \rangle$. The deadline is computed by the GETDEADLINE function of the host's schedule, which determines the latest time at which the host must leave the current location so as to have sufficient time to travel to the designated location of any previously made commitment in time. If the host has

no other commitments, this deadline value is infinity. This bid information is then added to a set. Once all the solicitations are considered, the TRANSMIT function sends all the bids to the coordinator.

Provisional Allocations and Reallocations. During the allocation process, two things happen in parallel—the allocation of tasks to hosts and the submission of bids by hosts. The process of bid submission is covered first followed by the allocation process. When a coordinator receives a bid, it is placed in a *bid list* that corresponds to the task that it was submitted for. The corresponding task is marked as “red” in the task list indicating that it needs attention.

The bids in each bid list are sorted in descending order of the capability fraction of the bid. The capability fraction indicates whether a host is specialized for the task or not. A “jack of all trades” would use fewer of its capabilities for a task than a host that is specialized for the task in question. Sorting the tasks in this manner biases the algorithm to choose more specialized hosts before choosing hosts with broader capabilities, the rationale being that it is desirable to have hosts with broader capabilities available for tasks which may not have specialized hosts. To break ties between bids, we use the average of ratios of the preceding and succeeding velocity to the maximum velocity, which indicates how good a fit the task is in the schedule of the host. Higher ratios indicate a more constrained time slot and therefore a better fit in the schedule.

When bids are initially inserted into the lists, they are marked as “black” indicating that the host that submitted the bid is not provisionally allocated to a conflicting task. In the future, as hosts are provisionally allocated for a task, the other bids belonging to the host that are in conflict (due to spatiotemporal constraints or schedule constraints) with that particular provisional allocation are marked as gray.

The allocation process iterates over the task list every p seconds and moves tasks to the *outstanding task queue* if they are either (1) marked red due to a new bid being submitted or (2) their earliest start time or host imposed deadline is within $minT$ (a parameter to the algorithm) seconds of the current time. The outstanding task queue is processed in parallel as follows (see Figure 1): The coordinator removes the first task from the outstanding tasks queue and looks at the bid list for that task. The first bid that is marked as black in

Given a set of sorted bid lists with bids B , an outstanding task queue O , a task list T , a minimum threshold $minT$, and re-evaluation period of n

```

ALLOCATE( $O, B, T, minT, n$ )
  while  $O \neq \perp$  do
    WAITONEMPTY( $O$ )
     $t \leftarrow$  REMOVEFIRST( $O$ )
     $bid \leftarrow$  REMOVEFIRSTBLACK( $B, t$ )
    if  $bid \neq \perp$  then
      if  $alloc[t] = \perp$  then
         $alloc[t] \leftarrow bid$ 
        COLORASGRAY( $B, host[bid]$ )
      else
        COLORASGRAY( $B, host[ $\text{MAX}(alloc[t], bid)$ ]$ )
        COLORASBLACK( $B, host[ $\text{MIN}(alloc[t], bid)$ ]$ )
         $alloc[t] \leftarrow \text{MAX}(alloc[t], bid)$ 
    if ( $(start[t] - \text{GETSYSTEMTIME}()) \leq minT$  or  $alloc[t].$ 
       $deadline \leq \text{GETSYSTEMTIME}()$ ) and  $alloc[t] \neq \perp$ ) then
      NOTIFYHOST( $alloc[t]$ )
    else
       $color[t] \leftarrow yellow$ 
      INSERT( $T, t$ )

```

Figure 1. The allocation algorithm

the bid list is the best qualified host that has no other conflicts. This host is provisionally allocated to perform the task, and all other bids submitted by the host that conflict with this allocation are marked “gray”. If the task under consideration already has a provisional allocation, the algorithm chooses the better bid using the same criteria that we use to rank bids. If there is a change in the provisional allocation, the conflicting bids are updated accordingly with the new bid’s conflicts being marked as “gray” and the old bid’s conflicts reinstated as “black”. At this point, the coordinator checks whether the current time is within some $minT$ of the earliest starting time of the task or whether the deadline set by the host submitting the winning bid has arrived. If either of these is the case, then it makes the allocation final by notifying the host of its newly allocated task. If the current time does not fall within the $minT$ of the earliest start time or the deadline has not approached, the coordinator marks the task as “yellow”, indicating that it does not need further attention at this time and re-inserts it into the task list. If no bid is available at the time of the initial evaluation, then the coordinator re-inserts the task

The above scheme ensures that tasks are considered and re-considered by the allocation process every time a new bid comes in, if their start time is approaching, or if the deadline imposed by the host with the winning bid is approaching. This ensures that the allocation decision is made as late as possible and is the best

option available at the time. Note however, that we do have a greedy element in our approach which is that if a host submits a bid and the deadline imposed by that host is approaching, then we commit to that host rather than risk waiting for another, potentially better host to come along. The complexity of the algorithm is shown in Figure 2.

Accommodating Physical Mobility. To accommodate mobility, we must consider it when allocating tasks to hosts because it affects the ability to transfer results after it has finished the task to the host(s) that have been assigned subsequent tasks in the workflow. The preferred method is to transfer the results directly to the intended recipient via a publish-subscribe based protocol described in [19]. However, this may not always be possible due to the lack of a disconnected route [13] (a spatiotemporal series of store and forward hops) between the two hosts in question. In such cases, the source host can attempt to transmit the results to the coordinator using the same publish-subscribe based protocol. If this too is not possible, the source host must physically return to the coordinator and transfer results. The coordinator, once it receives results, stores them until the recipient of those results is within range and then transmits the results to that host. Since we cannot know of the existence of disconnected routes between hosts *a priori* without knowing their motion profile [9], in our allocation planning we always assume that the worst case scenario will occur, i.e., the host will need to return to the coordinator. To factor this additional travel during the allocation process, each host checks whether it has sufficient free time to bring the results back to the coordinator after completing the task without infringing on any other commitments. If the check is positive, the bid is added to the list, otherwise the bid is considered invalid and is not added. In this way, we ensure that even while hosts are physically mobile, there is a reliable way for them to exchange data.

Distributing the Allocation Process. The transition from a centralized allocation algorithm to a distributed one is made possible by using multiple coordinators to allocate a workflow. This transition requires two key changes: (1) splitting the workflow into discrete pieces and (2) modifying the behavior of the coordinators. These are described in detail below.

Dividing the Workflow among Coordinators. By

definition, the workflow for any activity is a monolithic entity. We assume that initially, a single coordinator has the specification of this monolithic workflow. We refer to this coordinator as the *initiating coordinator*. The initiating coordinator is responsible for fragmenting the workflow using either the k-min-cut or the geographic cut approach. If the k-min-cut approach is used, fragments are assigned randomly to the coordinators. If the geographic cut is used, fragments are assigned such that the coordinators are responsible for allocating tasks in the geographic area in which they operate. Each coordinator, in addition to receiving a fragment of the workflow also receives a table of tasks in the workflow that are not in the fragment allotted to it, along with the name of the coordinator responsible for assigning each of those tasks. Once a coordinator receives its fragment of the workflow, it can immediately begin executing the core allocation process as described earlier.

Changes in Coordinator Behavior. From the coordinator's point of view, the transition from a single coordinator to multiple coordinator requires only two relatively minor changes. The first relates to the bid submission process. If a coordinator has a bid from another host that is better, it immediately rejects the host's bid, which allows it to leave the locality (described in the next paragraph). Second, when calculating the travel time of hosts to return results to the coordinator, the coordinator now has to be aware as to whether the subsequent task that the results are destined for will be allocated by itself or another coordinator. If the task is allocated by the same coordinator, there is no change from the case of the single coordinator. If the task is allocated by a different coordinator, the destination of the travel is changed to be the coordinator that is allocating the subsequent task.

Controlling Host Motion. In a mobile setting, especially that of a MANET, where hosts are physically mobile, the effectiveness of the allocation process depends heavily on host motion. If hosts move in an adversarial manner, they can ensure that very few tasks get allocated. Even if hosts are not adversarial, e.g., if the hosts exhibit random movement, they can still affect the performance of the allocation process. Simply put, when hosts move randomly, it becomes a matter of chance whether they come within range of the coordinator and have tasks allocated to them. We found

that by imposing minor constraints on host behavior, it is possible to make the system more consistent and reliable.

The constraints we impose on host motion are as follows. When a host has time slots in its schedule that are free, it gravitates towards the *nearest* coordinator. Upon coming within range of a coordinator, it checks the solicitations as before. However, if there are no tasks for which they are suited, it immediately leaves and goes to the next closest coordinator that it has not yet visited. If it does submit bids, the coordinator checks whether the bid is better than the current best choice. If not, the coordinator notifies the host immediately of the failed bid. This ensures that a host is not waiting at a coordinator while other coordinators have tasks that it could perform. These constraints are non-intrusive and in fact replicate the command and control structure of the collaborative activities to which we have targeted our work, i.e., it is analogous to a worker returning to his supervisor to be assigned additional tasks. As such, we believe such constraints are reasonable. The performance of our approach with and without these constraints is discussed in Section 5.

When the system is operating in a fully distributed manner, coordinators are usually stationary and responsible for a fragment of the workflow. Worker hosts gravitate towards a coordinator and remain there if there is a task that needs to be allocated that it can perform competently. If the task is allocated to the host, it leaves, performs the task and then returns to the coordinator. If the task is not allocated, the host may move to another coordinator in search of tasks. The process of searching for tasks and then performing them goes on until all the tasks are completed. At this point worker hosts have no more work to do and they gather around the coordinators. Coordinators then transmit a termination signal that indicates that its portion of the workflow has been completed and shuts down. Eventually all coordinators shut down, indicating to the hosts that the workflow is complete.

5 Implementation & Evaluation

We implemented our scheme within CiAN [19], our WfMS for MANETs. Further details can be found at mobilab.cse.wustl.edu/Projects/CiAN. We also conducted simulation experiments.

| Step | Complexity | Total |
|--|----------------------|------------|
| PREPROCESSING | | |
| Determining Parallel Tasks | $O(n)$ | |
| Fragmenting the graph - Graph traversal to make buckets - Choosing minimum cuts | $O(n + e)$ $O(n)$ | $O(n + e)$ |
| CORE ALGORITHM (on Coordinator) | | |
| Sorting Tasks by Utility with BubbleSort - Complexity is n^2 only when all tasks are in parallel with every other task (pathological case) | $O(n^2)$ | |
| Formulating solicitations | $O(n)$ | |
| Inserting bids into sorted buckets - Assume each host submits a bid for each task (pathological case) | $O(hn)$ | |
| Allocation and re-allocation - Each task is evaluated initially and re-evaluated every time a host submits a bid | $O(hn)$ | $O(hn^2)$ |
| PROCESSING BIDS (on Worker Hosts) | | |
| Analyzing solicitations + submitting bids - One solicitation per task in workflow | $O(n)$ | |
| Compute travel time for each solicitation | $O(n)$ | $O(n)$ |

Figure 2. Computational complexity

Experimental Setup. Our experiments were conducted using a custom simulator. The simulator defines an area which was divided into 100 x 100 grid squares with each grid square being equivalent to 5 square meters. An area of this size represents a large construction project or a toxic spill area. When 1 coordinator is used, it is placed in the center of the area. When multiple coordinators are used, the area is divided into $\sqrt{c} \times \sqrt{c}$ sub-areas (where c is the number of coordinators in use) and a coordinator is placed in the middle of each sub-area.

Host Generation. For each host, we selected a random number of capabilities from a list of all possible capabilities. We also generated a schedule for each host that was initially left blank, i.e., hosts had no external commitments in our experiments. All hosts were set to move at a uniform speed of 1 grid square per second and the communication radius of the hosts is set at 25 meters. While 25 meters is relatively low given the range of current 802.11g/n radios, we chose this figure since our target devices are likely to be power constrained and hence would operate their radios at lower power settings. Each host was initially placed at a randomly generated location within the defined area. We also included one other state variable within the host which is their status. A host's status at a given time can be idle, waiting at coordinator, travelling to task, or working on task. Hosts moved either: (1) randomly - where hosts that were idle moved in

a random walk until it encountered a coordinator by chance. There was no control to make the host move to another coordinator if it didn't find work at the first one it encountered – this scheme served as our baseline or (2) under control - where hosts that were idle moved systematically from one coordinator to another until it found work.

Workflow Generation. We used randomly generated workflows in our experiments. For each task, we first generated a unique task identifier for the task. This is followed by the picking a location randomly where the task is to be performed. We then picked a capability at random from the master list of capabilities which becomes the capability that a host must have in order to complete the task. The remaining pieces of the task specification, i.e., the start time, duration, and end time were left blank since choosing correct values for these depended on where the task ends up in the overall workflow structure. These were filled in after the graph structure has been generated.

The graph structure of the workflow is generated as follows. We pick a task at random and make it the root task. When possible, we pick whether the next structure will be a split or a join, e.g., at the root task, we cannot have a join so we must pick a split. We then pick a degree of divergence or convergence at random between 1 and 6 (1 representing a sequence and values greater than 1 representing a split) and add edges as appropriate. We then pick tasks at random and place them at the sinks of the newly generated edges, ensuring that we do not add anything that is a predecessor of the task at the source of the edge. This continues until all tasks are connected in a graph structure. Next, we generate time values for tasks. We choose a random start time and duration for the first task. To get the deadline for this task, we add the duration of the task plus a random *buffer* to the start time of the task. For each of its children, we then assign a start time that is a random amount of time after the deadline of the root task. Note that we generate this random time separately for each edge. The duration and deadline are generated in the same way as the first task. When tasks have several incoming edges, they use the deadline of the task that finishes the latest to generate a start time. The workflow is split at this stage for applicable experiments (see Section 4).

E1: Number of Tasks and Partitioning Schemes.

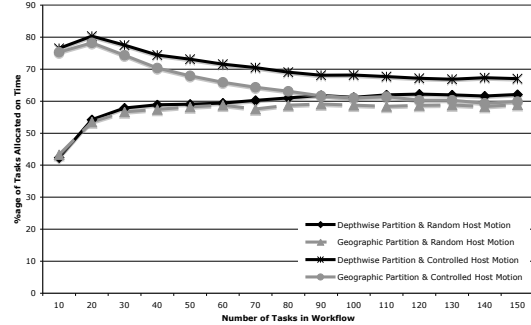


Figure 3. On time allocations

This experiment measures the effect of the number of workflow tasks on the percentage of tasks allocated. The results are shown in Figure 3. Each data point is an average of 10 workflows. Each workflow was allocated to between 10 and 100 hosts in steps of 10 using between 1 and 25 coordinators. Each point therefore is an average of 500 different runs. We observed the following: (1) When the workflows were partitioned by depth, on average 5.4% more tasks were allocated than if they were partitioned by geographic region when hosts exhibited controlled motion and 2% more than when host motion was random, all other things being equal. The reason for this was that in the depthwise partition, the distribution of tasks across coordinators was more equally distributed because the workflow was divided by the number of coordinators and contiguous pieces of roughly the same size went to each coordinator. In the geographic case, distributions were often skewed due to concentrations of tasks in certain areas, which led to starvation of hosts at some coordinators while other coordinators were temporarily overwhelmed. (2) For small number of workflow tasks, the controlled motion performed better as expected because it checks all coordinators systematically and the low number of tasks ensures that they all get allocated promptly. For workflows with greater than 80 tasks, we observed that performance was steady indicating that our approach scales well. The average on-time allocation percentage was only 71% but part of this was due to the way our system was set up. A discussion on this appears at the end of the section.

E2: Distributing the Allocation Process. An always reachable allocation engine has the advantage that it becomes a central point of coordination for the hosts and can result in reliable allocation of tasks. This was evidenced in our experiments where a single coor-

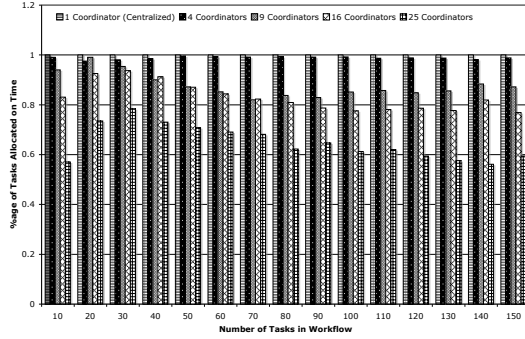


Figure 4. Effect of distributed allocation

dinator solution yielded 100% on-time task allocation under a variety of scenarios. The aim of this experiment was to study the deterioration in performance when the allocation process was distributed. The results are shown in Figure 4. For a given number of workflow tasks, we ran 10 different workflows with 100 hosts and between 1 and 25 coordinators. As can be seen, when 4 coordinators were used, over 98% of the tasks were allocated on time, with 9 and 16 coordinators 88% and 82% were allocated on time respectively. With 25 coordinators, the figure fell to 64%. For our largest workflow of 150 tasks, when 4 coordinators are used, each one has on average 38 tasks whereas with 25 coordinators each has only 6 tasks. Thus in the 25 coordinator case, when a host approaches a coordinator, it has very little choice of tasks to bid on and its likely that it would have to travel to many coordinators before it could find a task that it was suited to perform. However, with 4 coordinators, it would be much more likely to find a task at the first coordinator it visited. Excessive travel time with high number of coordinators negatively affects allocations.

E3: Controlling Host Motion. In an ideal situation, it is preferable that we do not impose any constraints on host motion for our allocation scheme to work effectively. However, with random motion of hosts, it becomes difficult to predict how the system will behave and offer any kind of guarantees. Hence, we decided to experiment with imposing certain minor constraints on host motion as discussed in Section 4. The results are shown in Figure 5. Each data point is an average of 150 workflows with between 10 and 150 tasks allocated to between 10 and 100 hosts. We observed that for low number of coordinators, the controlled approach worked better simply because it was

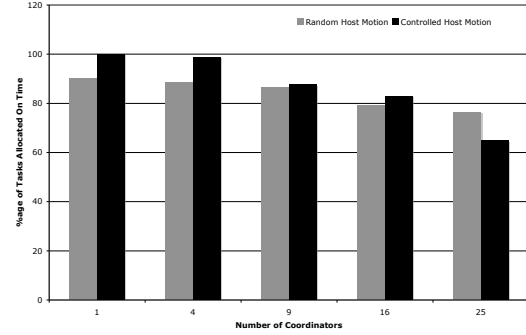


Figure 5. Comparing motion constraints

harder to find a coordinator by chance. However, when the area was saturated with coordinators, the advantage of the controlled approach disappeared. In fact, the systematic visiting of each and every coordinator in turn started hurting the controlled scheme because the random scheme could essentially visit coordinators in any order and pick up tasks more effectively. This shows that the controlled approach is effective in large areas with low number of coordinators whereas the random approach dominates in situations that have a small area and a large number of coordinators.

Discussion and Summary. The overall on-time allocation percentage of 71% at first glance appears poor but this is due mainly to the setup of our experiments. We chose a more rigorous approach where each task had a precisely defined start and end time. This is more strict than most workflow specifications that only impose a partial order among tasks. In fact, these hard timing constraints were the cause of most of the delayed allocations. In all our experiments, every delayed task was eventually allocated within a short period after its start time leading us to conclude that if we use only a partial order, our allocation rates will be very close to 100%. On the other hand, it is hard to work with spatiotemporal constraints when task times are not precisely specified. Finding a balance between the two remains an open challenge. Another recurring phenomenon we observed was that different combinations of coordinators, hosts, workflow tasks, patterns of motion, and manners of workflow fragmentation worked best with very different variations of our allocation scheme. This leads us to believe that it is perhaps desirable to customize an allocation process to the domain specific requirements as a more general algorithm is likely to perform much worse. This in-

vestigation is part of our future plans. In summary, we list the three high level inferences from our work: (1) a distributed allocation performance is close to a centralized approach as long as the degree of distribution is low, (2) motion constraints on hosts are useful in situations where task and coordinator density is low, and (3) an equal distribution of tasks across coordinators helps increase the overall number of tasks allocated.

6 Conclusion

Due to the volatility of MANETs and the lack of centralized infrastructure, it is desirable to have key processes within a WfMS operate in a distributed fashion. We have described a process by which a monolithic workflow is divided into smaller pieces and then allocated in a distributed fashion by multiple coordinators in a MANET setting. Our approach combines a bidding scheme with measures of utility and fitness to make allocation decisions. Our experiments indicate that our approach works well when the workflow is split into a reasonable number of sub-workflows and allocates about 70% of the tasks on time. If minor constraints on host motion are applied, our algorithm eventually allocates all tasks in all situations. We hope to build on this baseline work and increase the effectiveness of this approach in our future investigations.

References

- [1] ActiveBPEL engine. <http://www.active-endpoints.com/>.
- [2] Bpel v2.0. <http://www.oasis-open.org/>, 2006.
- [3] M. Corp. The biztalk server. <http://www.microsoft.com/biztalk/>.
- [4] M. Corp. Groove virtual office. <http://www.groove.net/home/index.cfm>.
- [5] T. S. Dahl, M. J. Mataric, and G. S. Sukhatme. Adaptive spatio-temporal organization in groups of robots. In *Proc. of IROS*, pages 1044–1049, 2002.
- [6] A. G. et al. Heuristic methods for solving job-shop scheduling problems. Technical report, Universidad Politécnic de Valencia, 2000.
- [7] G. H. et al. Sliver: A bpel workflow process execution engine for mobile devices. In *LNCS*, volume 4294, pages 503–508, 2006.
- [8] M. M. et al. Workpad: an adaptive peer-to-peer software infrastructure for supporting collaborative work of human operators in emergency/disaster scenarios. In *Proc. of CTS*, May 2006.
- [9] R. S. et al. Knowledge driven interactions with services across ad hoc networks. In *Proc. of ICSOC*, pages 222–231, 2004.
- [10] H.-L. Fang, P. Ross, and D. Corne. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In *Proc. of ICGA*, pages 375–382, 1993.
- [11] B. P. Gerkey and M. J. Mataric. Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5), October 2002.
- [12] B. P. Gerkey and M. J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, pages 939–954, September 2004.
- [13] R. Handorean, C. Gill, and G.-C. Roman. Accommodating transient connectivity in ad hoc and mobile settings. In *Proc. of Pervasive*, pages 305–322, 2004.
- [14] N. Liu, M. A. Abdelrahman, and S. Ramaswamy. A multi-agent model for reactive job shop scheduling. In *Proc. of the 36th Southeastern Symposium on System Theory*, pages 241–245, 2004.
- [15] G. Mainland, D. C. Parkes, and M. Welsh. Decentralized, adaptive resource allocation for sensor networks. In *Proc. of NSDI*, pages 23–33, 2005.
- [16] P. H. Mauguire, J.-C. Billaut, and J.-L. Bouquard. New single machine and job-shop scheduling problems with availability constraints. *Journal of Scheduling*, 8:211–231, 2005.
- [17] S. Muller-Wilken, F. Wienberg, and W. Lamersdorf. On integrating mobile devices into a workflow management scenario. In *Proc. of DEXA*, pages 186–190, 2000.
- [18] L. E. Parker. Alliance: An architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.
- [19] R. Sen, G.-C. Roman, and A. Frank. Cian: A language and middleware for collaboration in ad hoc networks. Technical report, Washington University in St. Louis, 2006.
- [20] H. Stormer and K. Knorr. Pda- and agent-based execution of workflow tasks. In *Proceedings of Informatik 2001*, pages 968–973, 2001.
- [21] S. Thatte. Xlang: Web services for business process design. http://www.gotdotnet.com/team/xml/_wssspecs/xlang-c/default.htm, 2001.
- [22] S. Yang. Job-shop scheduling with an adaptive neural network and local search hybrid approach. In *Proc. of IJCNN*, pages 2720–2727, 2006.
- [23] R. Zlot, A. Stentz, M. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *Proc. of ICRA*, 2002.