A Decision-Theoretic Approach to File Consistency in Constrained Peer-to-Peer Device Networks

David L. Roberts Sooraj Bhat Charles L. Isbell Jr.
Brian F. Cooper Jeffrey S. Pierce
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0706
{robertsd,sooraj,isbell,cooperb,jpierce}@cc.gatech.edu

ABSTRACT

As users interact with an increasing array of personal computing devices, maintaining consistency of data across those devices becomes significantly more difficult. Typical solutions assume either access to centralized servers, continual connectivity, or unbounded storage and CPU capacity. In practice, users own devices with widely varying processing and storage capabilities that use intermittent or sparsely-connected networks and incur (often asymmetric) transfer costs. We identify the conditions that enable the seamless management of a user's data across devices and present a multi-agent system built upon a decision-theoretic approach to constructing and executing multiple plans to achieve consistency in a peer-to-peer, partially observable, non-deterministic environment. We analyze the performance of these plans in comparison to a standard epidemic replication algorithm used in many database consistency applications.

1. INTRODUCTION AND MOTIVATION

Business professionals and academics regularly use multiple computing devices including desktops, laptops, PDAs, and cell phones. Each device is potentially powerful enough to provide the ability to view and manipulate arbitrary data; however, this power comes with the price of maintaining data consistency across each of those devices. For example, it is quite common for a user to edit a file on a desktop at work during the day (such as a paper they are composing) and then email it to themselves so they can continue writing when they get home at the end of the day. When they are finished for the evening, they will often email the latest revision to themselves again so they can print and revise it at work. Even if the user remembers to send the file, they will end up with multiple versions of the same file which increases the potential they will edit the wrong version and then have to spend time reconciling the differences. This problem is often exacerbated by the fact that the user may use more than two computers. For example, the user may have a desktop at work and at home as well as a laptop and PDA.

In this paper, we consider the problem of automatically and transparently ensuring that the user's data remains consistent across all devices they use. We construct a *Personal Information Environment* (PIE) representing a virtual data space not tied to any one physical device. The goal is to provide the user the illusion that every file is available on every device all the time.

To maintain consistency, users could try to use a single storage unit (such as a USB key or iPod) to store their files, mounting it on any device they are currently using. This would simplify the data synchronization problem as the master copy of all files would always be on the storage unit; however, there are several problems with this approach. First, it requires the user to carry the storage unit with them, and plug it in to every device they use. If the user forgets to plug it in, or leaves it somewhere, they will be unable to access their files. Second, these devices often have limited storage: a USB key only has 512 MB or 1 GB and the latest cell phones have only 2GB of storage[1], meaning that only a subset of the user's files may fit on the storage unit. Larger devices (such as iPods) are more expensive and likely to remain so for the foreseeable future. Third, the storage unit must be pluggable into any device that the user accesses. While most modern PCs have a USB port, few PDAs or cell phones do. Moreover, USB is slow; a user accessing a large amount of their data would need a faster interface (such as FireWire) but even fewer devices have those ports. Fourth, a user may access multiple devices at once. For example, a user may use a laptop and a desktop at the same time in their office, or may be sitting at a PC but also accessing their PDA. However, the storage unit can only be plugged into one device at a time, meaning the user's files are only accessible on one device. For all of these reasons, relying on a single physical storage device to manage user files often simply will not work. A software solution, such as the one presented here, can deal with each of these issues: it can always be running on every device, and it can use the full storage capacity of the device.

Another alternative is for the user to synchronize their files with a central file server (such as a CVS server, Samba server or NFS server). This approach avoids the physical limitations of a single storage unit. However, the central file server also faces its own problems. First, for many devices connectivity is intermittent. PDAs may only be connected when within BlueTooth range, while laptops may only be connected when near a WiFi hotspot.

[1]Even if one believes that 2GB provides ample storage, there is still an issue of limitations due to power consumption, something that is not keeping pace with growth of storage capacity.

Therefore, the device may not always be able to connect to the file server on demand. We might ask the user to monitor connectivity and ensure the device is connected when it is time to transfer files, but this places a burden on the user. Second, not everyone is willing to set up and maintain a centralized, always connected file server. Even users with broadband connections are usually behind a firewall, making it difficult for any device that they set up in their home (or work) to serve as a file server for when they are not at home (or work). We might ask the user to contract with a commercial storage service that charges a fee to act as the user's file server; however, many users are unwilling to trust their personal data to a commercial entity. Users may also prefer a free solution to a fee-charging one. Moreover, such a commercial service will have to be accessed over the internet, which is often much slower than a short range connection.

When we move toward a decentralized automated solution, several difficult problems arise. We must know the files the user will need on any particular device. The topology of the device network is not known and may change as devices come and go. As noted above, regular connectivity is not guaranteed, and for many users there is no centrally available server. Further, devices may not have a fixed IP address (*e.g.*, machines on consumer DSL networks may have continually changing IP addresses) or reside behind firewalls. As a result certain devices may never communicate directly with each other, making for a partially observable environment. In this situation, the role of limited capacity devices such as PDAs and cell phones frequently on a user's person becomes that of a crucial vessel to transfer data. Still, the computational and storage capability of each device varies considerably, as does the capability and cost of transferring data. Additionally, transferring data is non-deterministic. Executing, or planning to execute, a transfer between two devices does not guarantee that the data will transfer successfully even if those devices connect. In short, we have a problem of propagating information in a resource-constrained, distributed and peer-to-peer non-stationary network that is only locally observable.

Described this way, the problem seems hopeless; however, we have several advantages. First, users are often at least locally predictable. They cannot be in many places at once, allowing time to move data between devices. Even devices that are never directly connected (*e.g.*, an office machine behind a firewall or turned off at night and a home machine turned off during the day) can take advantage of intermediate devices to communicate (*e.g.*, a PDA or cell phone). Finally, because a limited number of files are touched in a given day and the number of devices is quite small, the problem admits practical computational tractability.

Our approach is for each device to identify important data that it might possess, other devices that might need that data, and to construct *plans* to move that data using whatever intermediaries may be necessary.[2] In addition, each device adapts the plans of other devices. Acting on a plan only requires that a device transfer data based on the utility of the plans associated with that data. Therefore, plans can be acted upon even by devices with very limited computational ability. Severely limited devices (*e.g.*, USB keys) can even contract out the construction and execution of plans to other more powerful devices in the network.

[2]Note that our goal is not to have every device have all data. First, each device should have the data that the user needs for it to have. Second each device should have any other data that helps other devices to have the data the user needs them to have.

In the rest of this paper, we describe in detail our algorithm for building and acting on collections of plans. We discuss several practical issues that arise and elucidate the assumptions that allow the system to work. Additionally, we compare the performance of our algorithm to that of the standard epidemic algorithm and point out the improvements in performance and behavior. Finally, we conclude with a discussion of our results and suggest future work.

2. RELATED WORK

One of the best-known systems for managing information on mobile devices is Coda [13]. Coda uses manually constructed or heuristically derived "hoard profiles" that determine which files should be replicated to which devices. Coda provides support for replicating a file between a device and a central file server, but does not support multi-hop transfers over resource-constrained devices. Further, Coda assumes that when a device is connected to the server, the connection is of sufficient duration and bandwidth to perform all necessary transfers. Other prominent mobile systems projects assume either a central file server, or high-bandwidth, always-on connections between devices.

There has been a large amount of other work on update consistency in disconnected or weakly connected networks. Examples include FICUS [18], Little Work [12], Bayou [7], and Thor [16]. The goal of most of these systems is to manage updates, propagating them to a primary copy or to various clients. Our goal is somewhat different: to place cached copies of files where a user will need them as proactively as possible. Once the replicas are placed, standard techniques can be used or extended to manage consistency.

Our work is based on ideas drawn from both multi-agent planning and decision-theoretic planning. While we do not explicitly represent cooperative or collaborative goals like some multi-agent systems, the agents in our system operate in unison to achieve a globally useful—if not globally consistent—state. In the context of agent collaboration, our agents have an implied contract to make their best effort to complete any and all multi-plans they receive during transfers.

In [9, 8], the authors present a framework for agent collaboration. They describe conditions necessary for agents to collaborate to achieve a single shared goal. [21] outlines a more complex architecture for multi-agent interactions, allowing for subgoal contracts to other agents, plan sharing, centralized plan storage, multi-technology integration, and other forms of agent interaction. There are also STEAM agents [20], built upon the *joint intentions* framework [15]. The implied contract between agents in the joint intentions framework requires that each agent, upon obtaining knowledge of another agent's goal, assumes the goal as its own, pursuing it until it is achieved or becomes irrelevant. STEAM agents propagate goal existence information and goal status information to each other via peer-to-peer updates.

In our domain, there is a shared goal that is not explicitly represented to the agents. As we shall see below, goals are represented as "multi-plans". These multi-plans are sent from agent to agent along with whatever data they were constructed for. Each device in the PIE is an element of a set θ of agents and every multi-plan can be thought of as an unsatisfied goal p. Therefore, the agents and all of the multi-plans currently being executed by the devices in the PIE represent a set of joint persistent goals $JPG(\theta, p_i)$. As the multi-plans are transferred from device to device, they are merged and updated with other multi-plans to reflect (as accurately as pos-

sible) the current status of the JPG. A multi-plan can become invalidated when a device receives information that there is a newer version of the data associated with that multi-plan. This is identified during the merge process and can trigger replanning if necessary.

Because the devices that comprise a PIE have varied characteristics, it is important for agents to have some notion of the ability of their peers. [1] presents a set of criteria for describing heterogeneous agents in complex planning architectures. It contains metrics that indicate characteristics of an agent's abilities and intentions. These metrics consist of *commitment* (a measure of an agent's desire to achieve a goal), *responsibility* (a measure of how much the agent must plan to see a goal reached), *authority* (ability of the agent to access resources), and *independence* (ability of the agent to construct plans). Agents in our system are all committed to joint goals; however, different types of devices in a PIE have varying levels of responsibility, authority, and independence. For example, a relatively powerful desktop computer may have high responsibility because it is almost always connected and has a fast processor that enables it to construct multi-plans on behalf of other agents; high authority for many of the same reasons; and high independence. On the other hand, a USB key will have little responsibility, authority, and independence. Using these measures, we tag devices with profiles that enable more powerful devices to aid in operations that less powerful devices may not be capable of.

There has been a great deal of effort to improve the performance of decision-theoretic planning by reducing the state-space through abstractions. For example, in [3] and [5] the authors present an abstraction method that reduces the state space to significantly improve the performance of constructing decision-theoretic plans. They prove a bound on the loss of plan precision.

By contrast, our approach exploits the fact that our state space is limited by the number of devices comprising a typical PIE. We can thus utilize a brute force greedy approach much in the spirit of early decision-theoretic planning systems such as [11] and [22]; however, we do improve over the limitations of those systems (suggested by [2]) by utilizing innovative utility functions to encode more complex information about the environment.

[10] defines a middle ground between traditional goal-directed agents and preference-based agents. In this middle ground the authors are able to define various stages of goal satisfaction, including preference (or lack thereof) for partially satisfied goals and the trade-off between satisfying a goal and the cost of resources consumed in service of the goal. Our characterization of expected plan utility is amenable to adjusting specific representations of action utility, action cost, and the probability of action success to encode each type of goal-directed behavior.

3. KEY CONCEPTS

In this section we lay the foundation for the Multi-Plan algorithm. We define several terms and derive the expected utility of plans and multi-plans. We conclude with a brief discussion of important implementation details.

3.1 Definitions

Our goal is to place data, here represented by files, on different devices. A *placement* is an ordered list of unique device IDs on which a file is to be located. For example, $l(1, 2, 3, 4)$ is the placement corresponding to a file being located on devices 1, 2, 3, and 4. $l(1, 3, 2, 1)$ is not considered because it corresponds directly to $l(1, 3, 2)$.

A *plan*, P_i, is an ordered list of placements: $\{l_{i,1}, l_{i,2}, ..., l_{i,k}\}$ where $l_{i,j}$ denotes the j-th placement of plan P_i. We constrain each plan so that each of its placements is a prefix of the placement that follows. For example, $P_i = \{l(1), l(1, 4), l(1, 4, 2), l(1, 4, 2, 5)\}$ is the three step plan moving some file from device 1 to device 5 via devices 4 and 2, in that order. As such, a plan can be abbreviated by its final placement: $P_i \equiv l(1, 4, 2, 5)$.

Each placement has an associated *utility*; that is, there is some value to having a particular file on a particular device or set of devices. We define the utility of a plan from those placement utilities: $U(P_i, j)$ is the utility of the plan P_i at position j ($U(P_i, j) = U(l(i_1, i_2, ..., i_j))$ where $l_{i,j} = l(i_1, ..., i_j)$).

If $p_{i,j}$ is the probability of transferring some bits successfully from device i to device j, and $c_{i,j}$ is the cost of transferring those bits from device i to device j, we can calculate the *expected utility* of a plan in a straightforward way. For notational convenience, let $L(P_i, j) = \prod_{x=1}^{j-1} p_{i_x, i_{x+1}}$ be the likelihood of data reaching device i_j from device i_1 following plan P_i; let $C(P_i, j) = \sum_{x=1}^{j-1} c_{i_x, i_{x+1}}$ be the cost associated with data reaching device i_j from device i_1 following plan P_i; let $NL(P_i, j)$ be the net loss[3] of the plan P_i at its placement $l_{i,j}$; and let $NV(P_i, j) = U(P_i, j-1) - C(P_i, j-1) - NL(P_i, j)$ be the net value associated with data reaching device i_{j-1} from device i_1 but not reaching device i_j following plan P_i. The expected utility of a plan is:

$$\begin{aligned} EU(P_k) = &\quad EU(\{l_{k,1}, ..., l_{k,n}\}) \\ \equiv &\quad EU(l(k_1, k_2, ..., k_n)) \end{aligned}$$

$$= \quad (1 - p_{k_1,k_2}) * -NL(P_k, 2) \tag{1}$$

$$+ \quad \sum_{i=3}^{n} L(P_k, i-1)(1 - p_{k_{i-1},k_i}) * NV(P_k, i) \tag{2}$$

$$+ \quad L(P_k, n) * (U(P_k, n) - C(P_k, n)) \tag{3}$$

Expression 1 is the cost of failure of the entire plan scaled by the probability of it failing during the first step. Expression 3 offsets the utility of the success of the plan with the cost of its execution and scales it by the probability of success. Expression 2 is a similar computation to Expression 3 but it considers the effect of the plan failing at each intermediate step.

Plans do not occur in a vacuum. There may be many plans across different files. A *multi-plan* is simply a (possibly ordered) set of plans: $M_i = \{P_{i,1}, ..., P_{i,n}\}$. Making an assumption about independence, we define the expected utility of a multi-plan to be the sum of the expected utilities of the plans it contains: $EU(M_i) = \sum_{P_k \in M_i} EU(P_k)$.

From the perspective of an individual agent, we seek a locally optimal (according to expected utility) set of multi-plans to approxi-

[3]The net loss is some measure of the negative effects of not getting data to its desired destination. In our experiments, it is the utility not realized by failing to complete the plan: $NL(P_i, j) = U(\{l_{i,j}, ..., l_{i,n}\}) = U(\{l_{i,1}, ..., l_{i,n}\}) - U(\{l_{i,1}, ..., l_{i,j-1}\})$, where n is the number of placements in a plan. There are other possibilities as well.

mate the globally optimal set. Because multi-plans are constructed in a partially observable environment, it would be infeasible to consider all inter-plan interaction. Assuming independence allows concise and computationally tractable plan construction.

3.2 A Note on Probabilities and Costs

In the previous section we defined a notion of expected utilities derived from other values such as placement utilities, costs, and connection probabilities. It is reasonable to wonder how the algorithm will arrive at these numbers. We are in the process of integrating multi-plans into the Accord [4] middleware system we are developing. Using Accord, we can derive values from actual user data. Here we discuss some of our choices and other possibilities.

The utility of a placement for a particular file is based on the "affinity" of that file for each of the devices in the placement, typically implemented as a linear combination of total reads and writes for that file on each device. Transfer probability is the empirical probability of the success of transferring a number of bits given a connection. Both probabilities and affinity can be stored for windows of time during the day and/or days of the week, as the probability of a cell phone being near the office computer on a Monday at 10am might be different than at 8pm. Using this type of encoding, temporal extent can be added to expected utility calculations by combining probabilities of connection for each time window between the time of the construction of the plan and the desired delivery time of the data.

Similarly, costs can be related to the actual bandwidth consumption during a transfer and the amount of a device's resources that would be consumed receiving, storing, and/or sending a file. Regardless of the specific representation of probabilities, utilities, and costs, our algorithm will select a multi-plan to maximize expected utility.

It is important to note that the relationship between costs and utilities is crucial. In particular, the "units" of each should be consistent; otherwise, undesirable behavior will result. For example, if cost is a relatively low percentage of utility, it is likely that most or all plans will have high positive expected utility. In the best case, only a small subset of possible plans should be identified for inclusion in a multi-plan.

It is also worth noting that any information used to give value to the abstract concepts of probabilities, costs, and utility will likely be based on statistics gathered by individual devices in the PIE. Much in the manner of the joint intentions framework, in order for a device to have accurate enough information to build a multi-plan to get data to another device, it will either need to connect directly to it, or receive the information from another third party device. The third party may have received that information through direct connection or via another third party, and so on. Interestingly enough, the data can then be sent between devices as the payload of the transfers that are executed as a result of the multi-plans that are based on it.

4. MULTI-PLANS

In this section, we present the Multi-Plan algorithm and show how it can be used to intelligently transfer files between devices. Multi-plans are constructed using a simple greedy, but locally optimal strategy. Our approach is an approximation of a true globally optimal set of multi-plans; however, that can only be computed if the environment is fully observable.

Each agent in the system uses a straightforward strategy.[4] Identify every file it contains that it believes is the most recent version and that is out of sync with other devices.[5] For each such file, construct a multi-plan (Section 4.1) to transfer the file to any device needing to be updated. When a connection is established with another device, exchange and synchronize file location information and sets of multi-plans (replanning if necessary), and execute a transfer ordering (Section 4.2). Repeat as necessary.

4.1 Construction of Multi-Plans

Suppose it is necessary to transfer a file f from a device x to another device y. Further, suppose there are d devices in the PIE. This algorithm constructs multi-plans to achieve that goal:

CONSTRUCT-MULTI-PLAN(f, d, x, y)
1 $Set \quad S \leftarrow$ NIL
2 $MultiPlan \quad MP \leftarrow$ NIL
3 **for** $i \leftarrow 2$ **to** d
4 **do** $S \leftarrow S \cup \{$ All Plans of length i from x to $y \}$
5 **for each** P_k **in** S
6 **do** $u \leftarrow EU(P_k)$
7 **if** $u > 0$
8 **then** $MP \leftarrow MP \cup P_k$
9 **return** MP

Note that when there are no positive utilities, the algorithm will return an empty multi-plan because any action (or transfer) would be expected to result in a decrease in overall utility. It can be shown that any multi-plan constructed according to our algorithm will yield a maximum expected utility plan.

LEMMA 1. *Let f be some data to transfer, d be the number of devices in the PIE, x be the original location of the file, and y be the ultimate destination of the file. A multi-plan to transfer f from x to y constructed using this Algorithm will yield a maximum expected utility multi-plan.*

In the interests of space, we have stated this as a lemma and omit a formal proof. The proof relies on two observations. First, adding only plans with positive expected utility will maximize the utility of the multi-plan. While this can be proved rigorously, it intuitively follows from the definition of utility. Second, considering only plans of length $2, ..., d$ is sufficient. A plan of length < 2 is always satisfied, and any plan of length $> d$ does not add any utility.

4.2 Transfer ordering

Once plans have been constructed, they have to be acted upon. Acting on a plan is defined to be transferring the data associated with it and updating its progress.

Suppose a device x has a set of multi-plans S and connects to a device y.

EXECUTE-TRANSFERS(S, x, y)
1 **for each** MP **in** S
2 **do** $u \leftarrow 0.0$
3 **for each** P_i **in** MP
4 **do if** NEXT-HOP-IS(P_i, y)

[4]For computationally limited devices like USB keys, these steps can be contracted to a peer device.
[5]The Accord middleware can provide this information.

5 **then** $u \leftarrow u + \text{EU}(P_i)$
6 $MP.value \leftarrow u$
7 SORT-SET-BY-VALUE(S)
8 **for each** MP **in** S
9 **do** TRANSFER($MP, MP.data$)

The algorithm first tags each multi-plan with a value based on the total expected utility that is due to plans containing device y as the next hop. The set of multi-plans are then sorted based on this value to obtain the final transfer ordering. The result of this operation is to transfer the data that has the highest expected utility based on arriving at y first. This ordering allows the agent to make a decision about which files are most important to get to the currently connected device. As we shall see below, this algorithm provides the most benefit when either connectivity between devices or the capacity of any crucial device is limited and the decision-theoretic value of the multi-plans influence the agent to optimally prioritize the files with the highest expected utility.

When device capacity is limited, a device must decide which files and associated multi-plans to accept or reject during transfers. In our case, a limited capacity device always accepts newer versions of files it already contains, but when faced with a choice accepts new files only if the multi-plans associated with the new file have higher utility than the lowest utility multi-plans for files already on the device. We later refer to this as the *smart eject/reject policy*.

4.3 Implementation Optimizations

There are a few implementation details that enable faster performance. While each placement must be considered during multi-plan construction, possible placements change very slowly and are pre-enumerated before any connection. Similarly, utility calculation can occur offline and incrementally. In practice, the values of the costs and utilities are based on information gathered from other devices, so some recalculation must occur in response to new information. Unfortunately, this new information will only be obtained during a connection. Therefore, this recalculation needs to be as efficient as possible.

Information obtained during a connection about the location of a file in the system can have one of two effects. First, if the new information indicates that the file has arrived at a destination for which there is a pending multi-plan, the effect will be that the need for the plan will be eliminated. Second, the information can indicate that a device previously believed to have a file no longer does. In that case, a new multi-plan may need to be constructed. Fortunately, the number of devices tends to be small, so plan construction does not require a significant amount of time. In the former case, it is important to note that replanning after information is exchanged at the beginning of a connection period will quickly correct any inaccuracies caused by multi-plans being constructed with out-of-date information.

Also, while not explicitly stated above, we allow for short circuit evaluation of multi-plans. For example, if a plan from a device x to a device y contains intermediate steps a and b ($x \rightarrow a \rightarrow b \rightarrow y$) but x connects directly to y before a, the plan will be short circuited. Then, when replanning occurs during future connections, both x and y know if a or b still need a copy of the data for which the multi-plan was constructed and will act accordingly.

5. ANALYSIS

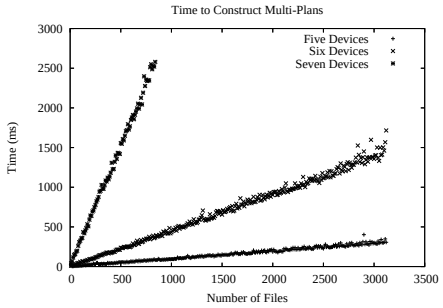

Figure 1: Time necessary to construct multi-plans. For each data set (d-devices), there were $\frac{(d-1)*f}{d}$ multi-plans constructed, where f is the number of files.

In order to analyze our algorithm's performance, we ran simulations for various device topologies, number of files, device capabilities, and transfer strategies. We ran simulations to determine the time required to plan, to measure performance during bootstrapping a PIE[6], and to determine performance under a typical workload.

5.1 Timings

In an attempt to identify what can be considered an upper bound on the offline CPU time necessary to bootstrap a PIE, we ran tests to determine how long it takes to construct multi-plans in environments with five, six, and seven devices. The results are plotted in Figure 1. All tests were conducted on a 1.7GHz Pentium M. The timings were gathered by iteratively increasing the number of files in the PIE. Each simulated device received $\frac{f}{d}$ simulated files to start the simulation where f is the number of files and d is the number of devices. The time to construct a plan to get each file to every other device in the PIE was then recorded and the process was repeated. In other words, each device constructed $\frac{(d-1)*f}{d}$ multi-plans.

Notice that for five devices and 3,000 files, 2,400 multi-plans can be constructed in less than 1/2 a second. Because the number of possible plans increases hyper-exponentially with the number of devices, it is not surprising to see the sharp increase in the time required to construct plans for six and seven devices; however, even those times are reasonable for a bootstrap process (*e.g.*, 1.5 seconds for 2500 multi-plans with 3,000 files and six devices).

It is worth noting that in practice, significantly fewer plans will need to be constructed during normal operations. Our initial user study has indicated that an average user modifies between five (during periods of normal use) and forty-five (during periods of heavy work such as coding or preparing a paper submission) files a day. Even in a PIE consisting of seven devices that connect to other devices once a day, constructing $45 \times 6 = 270$ multi-plans will take less than one second. With the exception of very extreme cases where connections between devices last for mere fractions of a second, time used to plan and replan during connections will be beneficial.

5.2 Bootstrapping

We ran simulations using five devices with 200 and 400 files in total. Files were distributed uniformly across the devices. Con-

[6]Bootstrapping a PIE occurs when devices (with possibly disjoint sets of files) are "introduced" for the first time and may exchange files with each other to achieve consistency.

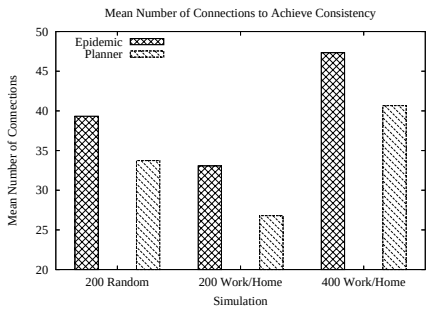

Figure 2: Multi-plans compared to the epidemic algorithm during the bootstrap process. Bars indicate the number of connections between devices necessary to achieve total consistency, averaged over 15 trials.

nections between devices were exclusive (*i.e.*, a device could not establish or accept a new connection while connected to another device) and intermittent (*i.e.*, were not always available and were randomly terminated after 3 to 10 seconds).

We ran three sets of trials to determine the performance of multi-plans in comparison to the epidemic replication algorithm [6] modeled after the epidemiological spread of disease, a popular mechanism for distributing data [17]. In the epidemic model, each device simply transfers any files missing from any other device it comes in contact with. For each set of trials, the simulations were started from the same initial state (*i.e.*, file locations, utility values, probabilities, costs, and so on). The simulations were considered complete when bootstrapping completed (when all devices had received a copy of every file in the PIE).

We chose to compare our work against epidemic replication rather than ad hoc network routing methods, such as those in [14], for a few reasons. First and foremost, our multi-plans are designed to provide data consistency by taking the path from a device to another device that balances the best chance for success with the most improvement along the way. While epidemic replication does not consider more than one hop at a time like multi-plans, it is designed to be used in situations where data consistency is the desired result. On the other hand, ad hoc network routing is designed solely to move information from one point to its destination as efficiently (in terms of bandwidth) as possible. This is not our goal. Secondly, there is a distinction to be made between peer-to-peer networks and peer-to-peer *device* networks. The former is a (usually) large number of loosely coupled and relatively powerful machines connected over the internet where decentralized and (more importantly) intentionally disjoint resources are shared. The latter is a small number of possibly resource constrained devices tightly (albeit intermittently) connected that are sharing data that is supposed to be consistent across all devices.

For our experiments, one set of trials was run with 200 files and a randomly generated connection scenario. The second set of trials was run using 200 files and connection probabilities that encode a typical five device scenario: there are two machines in the work place behind a firewall, two machines at home behind a firewall, and one device that is carried between work and home that is the only means of connection between the two sets of machines. The actual pairwise connection probabilities used are presented in Table 1. This scenario is a simplified version of the one described in

Connection Probabilities for the Bow Tie Trials					
Device	1	2	3	4	5
1	–	0.95	0.5	0.0	0.0
2	0.95	–	0.5	0.0	0.0
3	0.5	0.5	–	0.5	0.5
4	0.0	0.0	0.5	–	0.95
5	0.0	0.0	0.5	0.95	–

Table 1: Pairwise connection probabilities between all five devices in the bow tie simulation. While they happen to be symmetric in this example, they need not be.

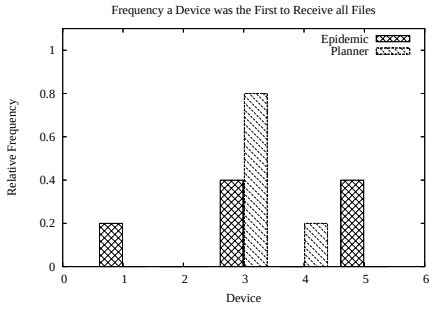

Figure 3: This plot indicates the relative frequency that a device in the simulation was the first to obtain a copy of all of the files during the bow tie simulations with 400 files.

the next section and we refer to it as the "bow tie" scenario because a graphical depiction of the connection scheme resembles a bow tie. The last set of trials was run with 400 files and the work/home connection scenario.

Figure 2 shows the performance of the two algorithms. There is a clear advantage to using the multi-plans in all cases. The multi-plans reached consistency with an average 14.25% fewer connections for the randomly generated connection scenario. Additionally, the simulations run using multi-plans outperformed the epidemic on the work/home connection scenario as well. The simulations run using 400 files yielded similar results to the random scenario with multi-plans reaching consistent file placement in an average of 14.08% fewer connections than the epidemic. Even more improvement occurs in the 200 file home/work scenario with an average of 18.95% fewer connections.

5.3 Behavior

Perhaps the most interesting result of the bootstrap simulations is the high level behavior seen during the bow tie simulations. By looking at the probabilities in Table 1, we know that any file on device 1 would have have to pass through device 3 before it can reach device 4 or 5. Figure 3 plots the relative frequency that a device in the bow tie simulation was the first to receive a copy of every file.

This is interesting to examine because the problem structure makes it necessary to get a copy of every file to device 3 before any of the other devices can obtain a copy. As can be seen from Figure 3, the multi-plan simulations far outperformed the epidemics in this sense. Device 3 first obtained a copy of every file in 80% of the simulations run using the multi-plans but only 40% of the simulations run using the epidemic.

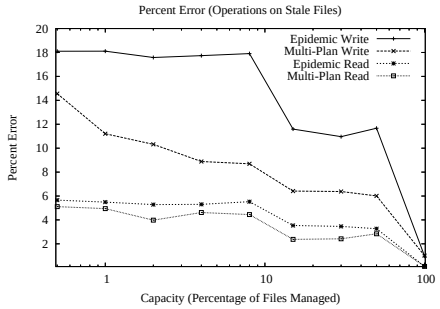

Figure 4: Multi-Plans vs. the epidemic algorithm as a function of "go between" device capacity.

5.4 Performance Under Load

Our interests are practical, so it is important to determine the performance of the multi-plans in a setting that more closely models everyday usage. Our simulator allows us to specify reads and writes on specific files over different periods of time, and to allow modifiable connection times. Further, to capture meaningful *error*, we measured *stale* accesses: the number of attempted reads and writes on files that are not the most recently modified version across all devices.

In each scenario, a user travels between home and work. There are two devices at each location and a fifth "go-between" device. Each location employs a firewall, so the go-between is the easiest way to transfer data between home and work. Scenarios differ in the connectivity and storage characteristics of the go-between: a BlueTooth phone (low capacity, low connectivity); a BlueTooth PDA (low capacity, high connectivity when docked); and a laptop (high capacity, mixed connectivity depending on connection mode).

All devices are capable of executing plans and using the smart eject/reject policy described earlier; however, when a cell phone is the go-between device, it is not capable of constructing plans. When the epidemic is used, there is no utility information available so the limited capacity device randomly ejects files to accept new incoming files.

Scenario data was gathered over 20 trials using 1000 files. Only 450 of the files were eligible for reads and writes by the user. The capacity of the go-between device was varied from 0.05% to 100% of the number of files in the simulation. For simulations with capacity under 8%, the go-between was modeled as a cell phone. For simulations between 8% and 50%, the go-between was modeled as a PDA. For the 100% capacity simulation, the go-between device was modeled as a laptop. Additionally, when the go-between devices was modeled as a cell phone, the frequency that a file was modified on it was reduced significantly in comparison to when it was modeled as a laptop or PDA. These variations allow us to test for different relationships between storage capacity, connection quality (probability of failure), and device usage.

For each scenario (and its variations), we simulated eight days worth of user activity. To gauge the adaptivity of each policy, we used a workload in which the working set of files changed after the first four days. This corresponds to a situation where the user begins working on one project on Sunday, finishes on Wednesday, and switches to a new project for the rest of the week. Regardless, portions of the working set remained in constant usage throughout the entire week, modeling files such as those modified by reading email.

We selected a workload of this nature for a few reasons. First, we feel it is an accurate model of typical users' file usage pattern. This type of workload is intended to model a scenario where a user is writing code for a few days to obtain results for a paper she is about to write. Once the code has been written, the user can "switch context" and begin writing the paper, effectively changing the working set of files. Second, we feel it is a fairly difficult scenario for our algorithm to handle and may provide an advantage to the epidemic algorithm. This advantage is due to the fact that the epidemic does not have a "memory" and likely will not be affected by the shift in working set. Our algorithm remembers the frequency of read and write operations on a file and uses that to determine the file's importance. When the working set changes in the middle of the simulated week, our algorithm has to choose between keeping the old working set of files consistent (as they seem important with a large number of reads and writes) and keeping the new set of files consistent (with only a few recent reads and writes).

The results are shown in Figure 4. Note that the percentage of stale write operations is higher in all cases than the percentage of stale read operations. This is an artifact of our decision to be consistent with our initial user studies and treat large sets of files in the simulation as read only files (*e.g.*, as would happen with music files).

With the exception of the 100% capacity simulations, the multi-plans (in most cases significantly) outperformed the epidemic algorithm. With severely limited capacity (0.05%), the multi-plans did not reduce the error by as large a margin as with other capacities but still provided some benefit. In the best case (with the go-between cell phone having 8% capacity), the error rate on write operations was less than half of the error rate for the epidemic model.

6. CONCLUSION AND FUTURE WORK

We have described the problem of managing data across an array of resource-constrained devices. Because it is difficult for users to deal with this problem directly, we have argued for building a distributed support system to allow multiple agents to act on the user's behalf.

The problem is complex and challenging. For example, knowledge of the working set of files is critical to ensuring that the proper files are prioritized for transfer in the presence of network connections with a high failure rate. Additionally, when certain devices never connect directly, it becomes even more crucial that there be some intelligent management of the routing of data via more highly connected devices.

We have presented motivation for the use of decision-theoretic multi-plans for file consistency management in a distributed peer-to-peer device network. We have shown that there is a distinct advantage to using planning techniques to intelligently order file transfers in the presence of constrained device networks over current algorithms for distributed data replication. Further, it is worth noting that in a scenario where every device has virtually unlimited storage and connectivity, our algorithm performs exactly as an epidemic replication algorithm would (within 0.115% error in simulation); however, when there is a physical constraint such as limited connection time or limited storage capacity, our algorithm far outperforms the epidemic (reducing error by as much as 51%).

Based on a series of initial user studies, we are currently working to fully integrate this method with our Accord middleware, a multi-device meta-data manager. The meta-data obtained using Accord will allow us to further augment our characterizations of utility and cost with other information such as locality and personal biases not easily identified from the type of raw statistics used in our experiments.

The end goal is to have systematic support for the use of statistical machine learning techniques to discover patterns in the usage of data and to infer relationships between resources for a particular user. For example, the context in which a file is being used—and therefore the other files it may rely on—can play an important role in its value. For example, suppose a user is writing a Java program. While the user is actually writing the code, it is important that all the Java classes in the project be up to date so compilation and testing will not fail; however, if the user has completed writing the code and is going to write documentation, it may not be quite as important that all of the files be up to date. We have some preliminary work on this topic [19]. Those behaviors and relationships can then be used to build personalized models of file affinity and to drive the construction of the most efficient multi-plans possible for any given user.

7. ACKNOWLEDGMENTS

We wish to thank Thomas M. Parks and Christopher Nevison for their donation of CPU time to gather data.

8. REFERENCES

[1] K. Barber and D. Han. Multi-agent planning under dynamic adaptive autonomy. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, San Diego, CA, 1998. IEEE.

[2] J. Blythe. Decision theoretic planning. *The AI Magazine*, 20(2):37–54, 1999.

[3] C. Boutilier and R. Dearden. Using abstractions for decision-theoretic planning with time constraints. In *AAAI'94: Proceedings of the Twelfth National Conference on Artificial Intelligence (vol. 2)*, pages 1016–1022. American Association for Artificial Intelligence, 1994.

[4] B. F. Cooper, C. L. Isbell, J. S. Pierce, D. L. Roberts, and S. Bhat. Accord: Middleware support for contextual, ubiquitous data management on user devices. Technical report, 2005. http://www.cc.gatech.edu/ cooperb/pubs/accord.pdf.

[5] R. Dearden and C. Boutilier. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, 89(1–2):219–283, 1997.

[6] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, , and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12, Vancouver, Canada, August 1987.

[7] W. K. Edwards, E. D. Mynatt, K. Petersen, M. Spreitzer, D. B. Terry, and M. Theimer. Designing and implementing asynchronous collaborative applications with bayou. In *ACM Symposium on User Interface Software and Technology*, pages 119–128, 1997.

[8] B. Grosz, L. Hunsberger, and S. Kraus. Planning and acting together. *The AI Magazine*, 20(4):23–34, 1999.

[9] B. J. Grosz and S. Kraus. Collaborative plans for group activities. In *Proceedings of IJCAI93*, Chambery, Savoie, France, 1993.

[10] P. Haddawy and S. Hanks. Utility models for goal-directed decision-theoretic planners. *Computational Intelligence*, 14(3):392–429, 1998.

[11] P. Haddawy and M. Suwandi. Decision-theoretic refinement planning using inheritance abstraction. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, University of Chicago, Illinois, 1994. AAAI Press.

[12] L. B. Huston and P. Honeyman. Partially connected operation. *Computing Systems*, 8(4):365–379, 1995.

[13] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, Feb. 1992.

[14] A. Kumar, J. J. Xu, and E. W. Zegura. Efficient and scalable query routing for unstructured peer-to-peer networks. In *Proceedings of 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1162–1173, 2005.

[15] H. J. Levesque, P. R. Cohen, and J. H. T. Nunes. On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 94–99, Los Altos, CA, 1990.

[16] B. Liskov, P. Johnson, R. Gruber, and L. Shrira. A highly available object repository for use in a heterogeneous distributed system. In *Proceedings of the 4th International Workshop on Persistent Objects*, pages 255–266, 1990.

[17] M. Rabinovich, N. H. Gehani, and A. Kononov. Efficient update propagation in epidemic replicated databases. In *Proc. of the 5th Int. Conf. on Extending Database Technology*, pages 207–222, 1996.

[18] P. L. Reiher, J. S. Heidemann, D. Ratner, G. Skinner, and G. J. Popek. Resolving file conflicts in the ficus file system. In *USENIX Summer*, pages 183–195, 1994.

[19] D. L. Roberts, C. L. Isbell, S. Bhat, B. F. Cooper, and J. S. Pierce. Decision making under dependent value constraints. Technical report, 2005. Submitted for publication to AAMAS06.

[20] M. Tambe. Agent architectures for flexible, practical teamwork. In *National Conference on Artificial Intelligence (AAAI)*, 1997.

[21] D. E. Wilkins and K. L. Myers. A multiagent planning architecture. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 154–163, 1998.

[22] M. Williamson and S. Hanks. Optimal planning with a goal-directed utility model. In K. J. Hammond, editor, *Proceedings of the Second International Conference on AI Planning Systems*, pages 176–181, Menlo Park, California, 1994. American Association for Artificial Intelligence.