Dexferizer: A service for data transfer optimization

Ercan Ucan and Timothy Roscoe
Systems Group, ETH Zurich

***Abstract*—**
We present an approach to optimizing the transfer of data objects within a user's collection of computers and personal devices, subject to a variety of user-defined quality metrics such as cost, power consumption, and latency. By abstracting object transfer as a high-level service, and employing declarative networking techniques to cast object transfer as a constrained optimization problem, we show how to transparently exploit techniques as diverse as swarming and multi-hop transfer through virtual machines. Using a data replication system as a driving application, we demonstrate that our approach can easily accommodate flexible policies not easily implemented with existing solutions, and can thereby result in savings in time, power, bandwidth, or other costs.

I. Introduction

This paper presents an approach to optimizing the transfer of data objects within a user's collection of computation devices, subject to a variety of user-defined quality metrics such as cost, power consumption, and latency. We use techniques from declarative networking together with application-defined policies to select appropriate transfer mechanisms and schedules.

Our goal is to evaluate the extent to which intelligent policy-based routing decisions, combined with a choice of different transport mechanisms, can outperform simple approaches such as centralized, cloud-based storage for a user's personal data. We are not the first group to propose a decentralized ensemble of personal devices (both physical and virtual) as an alternative to entrusting one's data to a large application provider; our focus here is on how to best use what connectivity exists between a user's computing hardware to manage their data.

In our target scenario, a user runs a personal replication system (along the lines of Cimbiosys [1] or Perspective [2] on a small (about 10 nodes) collection of devices such as phones, tablets, laptops, home PCs, and virtual machines and associated storage rented from cloud providers such as Amazon. The user generates new data items, by taking photos and videos, downloading music and documents.

The challenge is to transfer these objects between devices, satisfying user requirements for replication, availability, and privacy, while optimizing other user objectives such as latency, power consumption, and bandwidth cost for expensive links. Items in such an ensemble can vary in size from a few kilobytes up to a few gigabytes, and these items have varying synchronization and replication priorities. The problem is complicated by the limited resources on some nodes (such as phones), and the dynamic nature of connectivity.

In this paper, we present Dexferizer, a declarative transfer optimization service. Dexferizer employs a declarative approach whereby high-level requests are made on a *transfer service*, which schedules data transmission and late-binds the mechanism used for transferring an object, the paths taken by the data through the network, and the origin nodes of the data itself.

Furthermore, clients of the system write *transfer policies* in order to constrain how objects are transferred (such as avoiding public links or intermediate nodes, or transferring only at certain times). They also choose the metrics to be optimized for in planning the transfer (such as monetary cost, latency, etc.), and relative importance of particular object types from a transfer perspective.

The contributions of the paper are as follows: Firstly, we present declarative techniques to represent and reason about networks and optimize transfers using a constraint logic programming (CLP) approach. Secondly, we present an expressive and extensible grammar for declarative transfer policies. Finally, we show potential for bandwidth, time, cost, and power savings from late-binding transfers and incorporating user-selected transfer policies into the optimization process.

In the next section, we present the background work we build upon in our system, and then in Section III further motivate the problem using concrete scenarios. In Section IV, we present our model and general approach, and Section V presents the design decisions in the implementation of our research prototype. Our evaluation is present in Section VI, and we conclude and discuss future work in Section VII.

II. Background

Over 50% of today's Internet traffic comprises bulk data transfers [3]. Therefore, many research efforts have been made in order to increase the efficiency and speed of such transfers. DOT [4] is such an approach, in which the applications negotiate the transfer first, and the actual transfer is carried out subsequently by a separate transfer service. This way DOT enables the object transfers to be performed via different transport mechanisms without the applications having to know about the underlying details of the transport.

We build on DOT's concept of abstracting data transfers as single operations. However, in its current form, DOT does not aim to optimize the transfers based on changing network circumstances or user preferences, nor does it try to create an optimal schedule to carry out a set of transfer requests.

In a personal data management environment such as the one we target, a number of different resources can be considered scarce. Bandwidth may be limited. Energy is a valuable resource in mobile and hand-held devices in a personal network

978-1-4577-0103-0/11/$26.00 © 2011 IEEE

system. Furthermore, money is a constraining resource in such systems - there are financial costs as well as efficiency benefits from renting cloud storage or computation (including use of virtual machines as intermediate routers for data object transfers), and many mobile devices support transfer over cheap, but intermittently available, WiFi connections as well as expensive but more ubiquitous 3G data services.

Haggle [5], an architecture for seamless network connectivity in mobile environments, has also influenced our ideas. The key idea is to separate the application logic from the underlying networking technology, and delegate network decisions to a central resource manager running on each node in the network. This manager can select the right just-in-time bindings for the node's network interfaces and protocols, based on application-level semantics. Other projects have also exploited late-binding of interfaces to provide flexibility to applications. For instance, Horde [6] provides flexible network striping across different network channels, based on application-level policies. Wang et. al. [7] propose policy-enabled handoff across heterogeneous wireless networks, with a focus on limiting the need for user involvement in dynamic environments. We see our work as more data transfer-centric, and therefore complementary to these systems.

We also use ideas from late-binding naming approaches. The Intentional Naming System (INS) [8] uses a language based on a hierarchy of attributes and values for its names, which allows provider or consumer nodes to describe the services they provide or require. Applications benefit from INS's late binding support: the binding between the intentional name and the node location is established when the message is delivered rather than when the request is resolved.

The Unmanaged Internet Architecture (UIA) [9] is a distributed name system that aims to provide zero-configuration connectivity among different mobile devices through "personal names". Users own a local namespace shared among all their devices and available on every device, and assign personal names to each of their devices as well as other users, and access their friends' namespaces. Once assigned, these names remain persistently bound to their targets, regardless of their network location. These and similar mechanisms could use our work to benefit from reasoning about heterogeneous network elements and connectivity element.

Our work in this paper can also be viewed as an instance of cross-layer visibility [10]. The limited visibility across layers in a networked system hinders network management tasks, failure diagnosis, and ultimately the reliability of the network [11] (similar observations have motivated architectural work like Plutarch [12] and the Metanet [13]).

A rather different point in the design space is Mobile Ad-Hoc Networks (MANETs). MANET routing protocols [14]–[16] are decentralized and are designed to quickly react to device and link failures. Handling heterogeneity has not been a primary focus of MANET research, but we believe our approach can be extended to MANETs. Moreover, MANET protocols could also be expressed declaratively in our system, as with other routing approaches.

Finally, DTN [17] proposes a network architecture for challenged environments, and an API based on possibly-unreliable asynchronous message forwarding, with limited expectations of end-to-end connectivity and node resources.

III. Problem description

In this section we present a series of concrete scenarios demonstrating potential downsides of a naive implementation of a partial replication and file synchronization system that runs on a personal device ensemble.

A. Money is a scarce resource

Resources like metered 3G/2G data connections or virtual machines rented from cloud providers can incur significant monetary cost, which depends on usage.

For example, Swisscom currently offers [18] a 3G data plan at a monthly subscription rate of 45 CHF which has a data transfer cap at 250 MB, and charges 0.10 CHF per MB thereafter. Pricing on this plan for international roaming depends on where the user is connecting from. In Western Europe, a 24-hour slot costs 7 CHF with a cap of 5 MB. In Eastern Europe, a 24-hour slot costs 14 CHF with a 5 MB cap. For the rest of the world, each 30KB chunk of data costs 0.30 CHF [19]. As the numbers suggest, data transfer on 3G connections can become quite complicated and costly.

Consider a user taking photographs with the phone of about 700KB each, and suppose the default replication policy copies the photos to at least 3 other machines in the user's ensemble. A naive replication mechanism using 3G will transfer 2.1 MB to replicate one photo. This alone will exhaust the user's domestic bandwidth cap after 119 photos, and further pictures will cost 0.21 CHF each to replicate. At 5 pictures a day (150 a month), this leads to a monthly bill of 51.5 CHF only with photo transfers.

Of course, the user can turn off the synchronization system until they get home, and then transfer the photos via USB or WiFi, at substantial saving, but this prevents the immediate replication of other objects (such as important documents or critically important pictures). Synchronization systems like Dropbox [20] do not currently provide any way to prioritize replication by object type or attributes. Ideally, the user should maximize use of the 3G transfer without incurring extra cost, replicating the most important items first.

B. Time is a scarce resource

Consider a hypothetical scenario in the context of data replication where a user's home server crashes. The server contains a large number of replicas of items like photos and music files. Figure 1 shows the nodes in the user's replication system, and the network topology at the time of the crash. The replication system triggers an alert and decides to replicate data currently stored on the Desktop PC and the phone onto two more nodes: Amazon EC2 and GoGrid virtual machines. A naive approach can overload a single node or a single link for transferring data items to different targets, even though the items can be fetched from other nodes in the system as well. For example, the user may experience long transfer times (during which data durability is compromised) if the naive system decides that all the data items to replicate should be copied to EC2 and GoGrid from the phone.

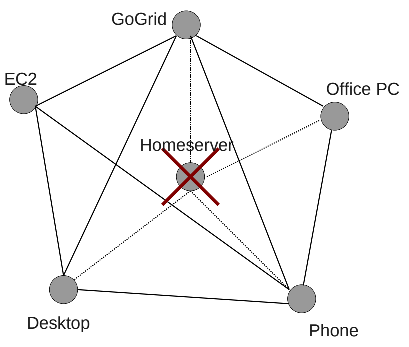

Fig. 1. Example transfer scenario in case of a home server crash.

C. *Energy is a scarce resource*

Energy is widely recognized as a scarce resource in handheld devices. We compiled the information shown in Table I from different usage scenarios presented in [21]. The table shows the power consumption (actually, current draw) of the Nokia N900 phone in various states, obtained using the onboard battery monitor. A 'T' in the 'Type' column means the data is the total energy consumption of the whole device, and a '+' means the data is additional energy consumed only by the device component.

An N900 on a 3G connection sending a file at 150 kbit/s draws 375 mA, and when receiving at 200 kbit/s draws 275 mA. This is more power than playing an MP3, activating the camera and showing a preview image, or even vibrating the phone continuously.

Device State	**Current**	**Type**
3G connection sending a file at 150 kbit/s	375 mA	T
3G connection receiving at 200 kbit/s	275 mA	T
3G connected, good signal, no data activity	8 mA	+
2G connected, good signal, no data activity	5 mA	+
Media player playing an MP3, one step above mute	110 mA	T
Camera active, showing preview, back-light off	210 mA	T
Vibration at max level	100 mA	+

TABLE I
Nokia N900 hardware power consumption data

It is clear that replicating data to or from a phone using a 3G connection will cost a lot of energy. Such devices can often use other connections (e.g. USB) that consume less power if they are available, and transfers of some (but not all) items can be delayed if it is known that these connections will be available later. Moreover, multicasting from such a device can significantly save energy.

IV. Approach

We now describe the problem in more detail and present our solution framework. Our problem representation consists of three parts: a network model, a model for user-specified transfer policies, and a model to express conditions on the optimal transfer schedule and mechanism. As described in Section V below, our representation is encoded using Constraint Logic Programming and, in particular, the ECL^{i}PS^e CLP engine based on Prolog.

A. *Network Model*

We represent the network as a set of links. These are not necessarily physical links, but represent the connectivity between pairs of nodes in the user's ensemble of devices. In addition, onto these links we map a set of transfer mechanisms.

1) Link quality information: Network links in our model are assumed to run over IP, and have the following properties:

- *type* : Type of a link (USB, 3G, Ethernet, etc.)
- *latency* : Ping latency between the end-points.
- *bandwidth*: Bandwidth estimation of the link.
- *load* : Current link load, estimated on a set of increasing timescales from seconds to hours.
- *cost* : Monetary cost of using the link. In practice, this is often not a scalar value but a function of volume and time derived from a pricing model.
- *power consumption* : Energy cost of using this link, again as a function of time and volume.

The precise representation of a link in Dexferizer is shown later in this section. Connectivity information is monitored and updated continuously and is used when making routing decisions.

2) Transfer mechanisms: Dexferizer can execute a transfer request using a variety of models, classified as follows:

- *point-to-point* : Transfer from a specific source to one destination node.
- *source-agnostic point-to-point* : Transfer from any possible source to one destination node.
- *swarming* : BitTorrent-like transfer mechanism where parts/chunks of a file can be gathered from multiple nodes via multiple links at the same time to a sink node.
- *multicast* : Transfer employing an application-level multicast tree.

Our goal in Dexferizer is to support all these different mechanisms, and select the optimal one based on network and system state, transparently to the application.

B. *Transfer Policies*

We also need a representation for transfer policies. We first describe some example transfer policy scenarios informally in English to illustrate a range of transfer policies, and then show how these are represented formally in Dexferizer as Prolog inference rules.

Our example scenarios are:

- *Constraints on the media type,* e.g. "never copy any of my video files over a 3G connection."

- *Constraints on single item size,* e.g. “never copy files bigger than 50MB over 3G.”
- *Time-based transfer size constraints,* e.g. “don’t copy more than 10MB per day over 3G.”
- *Item based prioritization,* e.g. “always synchronize my to-do list first.”
- *Privacy constraints,* e.g. “never synchronize my private photos through cloud resources.”
- *Link priority,* e.g. “always prefer USB or Ethernet to Bluetooth or WiFi.”

We express such policies in Dexferizer using CLP, and exploit Prolog predicates to abstract away from specific entities (items, links, nodes) in the system. We present some examples which illustrate how we can aggregate policies based on simple rules, starting with two predicates that classify files based on media type.

An object is a picture if it is of type `JPEG` or of type `PNG`, and is a video if it is of type `MPEG` or `AVI`:

```
picture_item(ItemID) :-
    item{itemid:ItemID, type:'JPEG'};
    item{itemid:ItemID, type:'PNG'}.

video_item(ItemID) :-
    item{itemid:ItemID, type:'MPEG'};
    item{itemid:ItemID, type:'AVI'}.
```

We also use a convenient predicate to refer to “any device”:

```
any_device(ResID) :- device{resid:ResID}.
```

All the data the system has about items, devices, and links are stored as Prolog facts. The next example shows how a picture called ‘bosphorus.jpeg’, created on 19 January 2011, and tagged “public” (i.e. not private) is represented:

```
item(photos/bosphorus.jpeg, 'JPEG', 1295448700, public,...).
```

Below we first show a phone called `nokiaN900`, which is mobile and owned by the user (as opposed to a rented virtual machine), and then a link in the network showing that the phone has an active 3G wireless connection to a desktop, with a latency of about 178 milliseconds; the bandwidth of the link is 150 kbit/s, and the current draw is 375mA:

```
device(nokiaN900, mobile, phone, owned,...).
link('nokiaN900','desktop', '3g', 178.742752, 150, 375, ...).
```

Transfer policies can now build on these twin concepts of facts and predicates about both devices and items. A transfer policy has the form:

```
  xfer_policy( IP, SrcP, DstP, TR, LP ).
```

`IP` corresponds to a list of item predicates, `SrcP` and `DstP` correspond to lists of device predicates. `TR` is the transfer relation which allows or disallows the transfer. `LP` is a list of link predicates. For example:

```
  xfer_policy([any_item], [phone_device], [any_device],
              [xfernone], [any_link]).
```

This expresses the policy: “Do not use a phone as a transfer source while replicating items”.

The next declaration expresses the policy “Never copy any of my video files over a 3G connection”:

```
  xfer_policy([video_item], [any_device], [any_device],
              [xfernone], [3g_link]).
```

Our optimization framework also allows prioritization of transfer requests. This can be done at the granularity of item predicates. Transfer prioritizations have the following form:

```
  xfer_priority([IPList, IPList, IPList, ...]).
```

As with transfer policies, IPList is a list of item predicates, and the predicates inside an IPList are treated as conjunctions. `xfer_priority` fact is a list of IPLists, and the ordering among these lists specifies the transfer priorities of corresponding items. For example:

```
  xfer_priority([[doc_item, private_item], [picture_item],
                [video_item]]).
```

This priority policy says that any document item tagged by the user as “private” should be treated with the highest priority in a transfer situation. Photo items are of the second highest priority, followed by the video items. Transfer of other items in the system will follow after these three item groups.

C. Formulating the optimization problem

We cast the task of providing data transfer as an optimization problem. When a transfer service is given a series of transfer requests to be performed, often there are multiple end-points (source, destination), paths, and mechanisms which could carry out these transfers in the network.

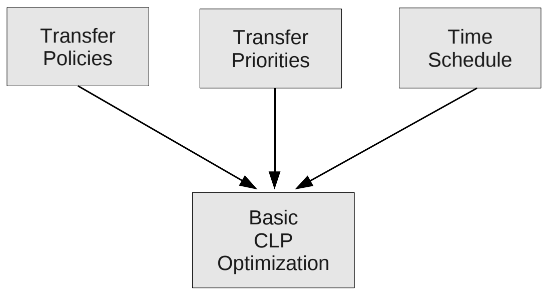

Fig. 2. High-level architecture of the Dexferizer

Figure 2 shows the logical structure of our problem formulation as a constraint logic programming optimization. In a nutshell, the basic optimization process examines possible transfer scenarios, decides how ‘good’ they are based on a cost or utility calculation mechanism, and selects the best possible solution. Transfer policies, transfer priorities and other types of declarative information (such as time schedules for transfers) are also incorporated into the optimization process, acting as constraints on the search space. In the following two sections we describe how we incorporate the potential transfer endpoints and routing information into the optimizer.

1) Incorporation of transfer endpoints information: We represent a transfer request as 3 two-dimensional matrices. Figure 3 shows an example: the columns of the matrices correspond to devices, and the rows represent data items in the system. Figure 3a is the *source matrix* and represents the current view of the replication system. Figure 3b shows the *destination matrix* representing the target configuration to be reached by the replication system. Figure 3c shows the *difference matrix* and is derived from the first two. It represents the actions to be performed in order to reach the desired state of the replication system.

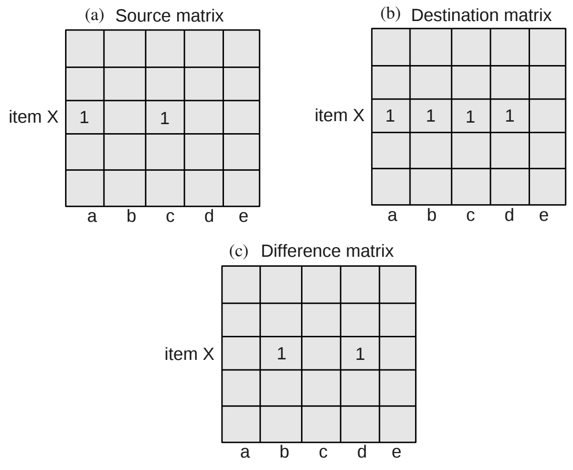

Fig. 3. Source, destination and difference matrices representing a request

As shown in Figure 3, item X has two potential sources (a and c), and two destinations (b and d). Potential transfer configurations can be written as $\{(S1, b), (S2, d)\}$ where $S1$ and $S2$ can be any non-empty subset of the $\{a, c\}$. $S1$ and $S2$ can be directly translated into Prolog variables.

Figure 3 depicts the case for a single file transfer. However, in real applications, the task of optimization should consider multiple transfer requests belonging to multiple data items at the same time. Therefore, in reality, the optimizer deals with a 3 dimensional matrix, which is a combination of 2 dimensional difference matrices for all the items to be transferred.

Treating the problem as an optimization requires a cost metric (or, equivalently, a utility metric) to compare potential solution scenarios. A cost function weighs potential transfer configurations and says how much a particular solution 'costs' the user, whereas a utility function says how useful a particular solution scenario is for the user. In our approach, depending on the desired criteria, the optimizer can be fed either type of function and told to minimize cost or maximize utility.

As an initial representation of a cost or a utility function, we use a 2-dimensional matrix where the rows represent the destination and the columns represent the source of a transfer. The value of element [x, y] of the matrix denotes the cost or utility of performing a transfer from a source node y to a destination node x. The matrix can be populated based on different optimization criteria. One advantage of this model is the ability to flexibly plug custom cost and utility calculation schemes into the framework. Here we describe two potential cost and utility functions that can be plugged into the formulation as the criteria of optimization.

1) **Network latency as cost:** One potential criteria for cost can be the network latency among the nodes between which the transfers are to be carried out.
2) **Bandwidth as utility / Transfer time as cost:** Bandwith utilization can be a criteria to maximize. This approach can also be represented as a cost criteria where the cost is the completion time of the issued transfer requests.
3) **Money as cost:** Another alternative for a cost criteria can be money, for example using pricing models for 3G connections and rented cloud virtual machines.
4) **Power consumption as cost:** Another optimization criteria can be the power consumption as it is an important concern for mobile and hand-held devices.

2) Incorporating routing information: The knowledge about and enumeration over the potential transfer *endpoints* alone is not always sufficient for optimization. There are scenarios in which the solution found by the optimizer would be sub-optimal; an example is illustrated in Figure 4. In this scenario, a file residing on EC2 and GoGrid nodes needs to be transferred onto the phone. The phone is connected to EC2 and GoGrid over a slow 3G link, and to the desktop via a USB cable. The desktop PC is connected to EC2 and GoGrid nodes over a fast internet connection. Here, looking at the bandwidth values the links provide, the optimal transfer situation would be that the file gets copied in parallel via the Desktop PC from EC2 and GoGrid machines over to the phone.

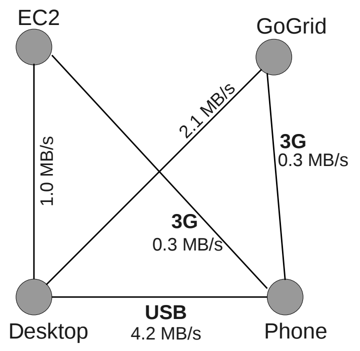

Fig. 4. A use case for supporting the integration of routing information.

This scenario demonstrates two dimensions of the problem space. The first dimension is the end points involved in the transfer of an object. The second dimension involves the possible paths the transfers can be carried out over, once the sources and destinations are determined. Considering only the potential transfer *endpoints* cannot completely capture the whole optimization space.

We capture the second dimension by adding reasoning about path and routing information into the optimization framework. The optimizer contains flexible and tunable, declarative routing algorithms that run using the existing declarative link and device information in the system. The optimizer can generate multi-hop routes for the transfers based on one of the various metrics such as latency, bandwidth, or monetary cost that it has been tuned for.

V. Implementation

The client application for Dexferizer is Anzere [22], a personal data storage and replication system implemented mostly in Python, currently containing more than 32,000 lines of code. One of the components of Anzere is the *overlay network*. This module contains *network sensors* that continuously monitor the network, link and device status, and generate up-to-date declarative system information regarding this network state. Similarly, the *storage* module of the replication system contains *storage sensors* that monitor the status of the data items in the system and generate declarative information about them.

Transfer scheduling and optimization code of Dexferizer is implemented entirely in ECL^{i}PS^{e} constraint logic programming language, a dialect of Prolog with constraint programming extensions. In the current design, the optimization code runs centrally on one of the devices in the ensemble. This device maintains complete metadata of the items, devices and links in its knowledge base and acts as a coordinator of transfer requests. The search routine used by the current optimizer is relatively naive: we use the `minimize` function inside the `branch_and_bound` library provided by the ECL^{i}PS^{e} CLP solver.

Figure 5 illustrates the logical stratification of declarative information and mechanisms in Dexferizer. At the base of the declarative information stack, there is the network and item related data as Prolog facts. One layer above, there are Prolog predicates written by developers. These are used to extract information about various properties of the data from the layer below. With the use of such predicates, the items, devices and the network resources can be reasoned about using their properties at a more abstract level instead of hard-coded names that they are attached to. Above this are transfer and priority policies, some of which will be written by users but most of which we anticipate being provided by developers in the form of a library.

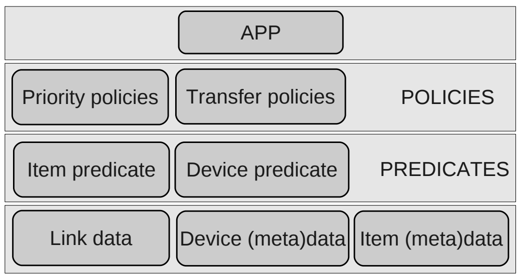

Fig. 5. Declarative transfer service abstraction layers.

Another key difference between traditional approaches and the way Dexferizer models the problem is the design of the communication scheme between the application and the transfer service itself. In Dexferizer, these two modules communicate over a declarative model. The transfer requests are represented in a fashion similar to a database, via the difference matrix. The database is examined by Dexferizer and the schedules of the transfers are generated after the incorporation of user specified transfer policies, network and cost models.

VI. Evaluation

In this section we present initial results on the prototype implementation of Dexferizer. Some of the figures we present in this section, while insightful, are still at an illustrative stage regarding the approach and the idea, rather than being exhaustive and comprehensive. We focus on the behavior of the optimizer and its tunability via different transfer policies. We also present initial numbers on how the optimizer scales with different numbers of items and in different transfer scenarios followed by results examining the resource consumption of the optimizer. We argue that our results imply significant potential savings on cost, time and power usage.

Unless otherwise noted, our experiment and simulation setup assumes a device ensemble consisting of 7 devices (desktop PC, office PC, laptop, smartphone, homeserver and 2 rented cloud virtual machines). We use different workloads for different experiments and we describe these in as necessary.

A. Comparison of optimized and naive approaches

We first compare the behavior of Dexferizer with that of naive approaches.

1) Baseline: optimizer set for maximum overall bandwidth utilization: In this baseline experiment, we compare the utility of the solutions generated using two different approaches. In the case of an unoptimized default behavior, the source node of a transfer is selected as the first node encountered in a scan over the nodes in the network, whereas the transfer optimizer considers different possible transfer scenarios and mechanisms, and optimizes the solution to obtain the highest utility value possible. We used a synthetic workload for this baseline experiment. Each item in the data set has up to 2 initial copies randomly distributed over all the devices in the ensemble. One new replica for each item is going to be created on one of the other devices (also randomly chosen) in the system.

Table II shows the utility matrix used in Figure 6. The columns are the potential devices (sources) that a file can be copied from. The rows are the potential destination (sink) devices for a transfer. The matrix cell (x, y) shows the utility of performing a transfer from the source node y to the sink node x. The numbers inside the cells of the matrix can be thought of as representative bandwidth (in megabytes/s) values in between devices.

We emphasize that these values are not completely realistic, but they currently act as representative placeholders to demonstrate the use of the utility matrix. In the actual system,

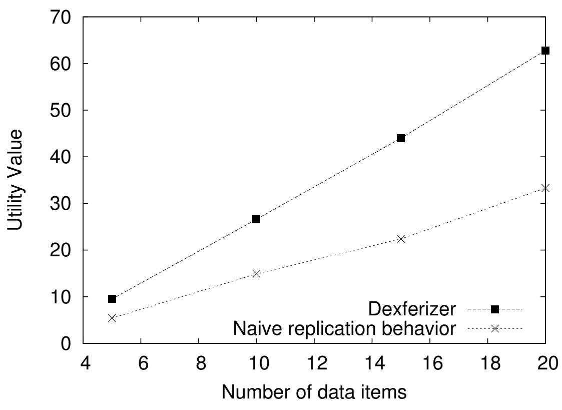

Fig. 6. Utility comparison between optimized versus naive behavior.

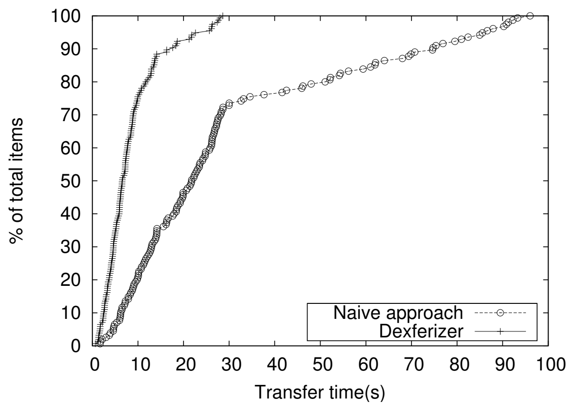

Fig. 7. Comparison of transfer times using Dexferizer and a naive approach.

the bandwidth is monitored continuously and the optimizer works with up to date information rather than the static and independent estimated bandwidth shown in this table. The aim of the optimizer is to maximize the utility, the overall bandwidth used in the network. It schedules the transfers to be carried out in a BitTorrent fashion wherever possible as this is the configuration that maximizes the utility. The naive approach does not show high utility since it always carries out the transfers from one node. This plot shows how the optimizer works with a utility function. The resulting meaning and the benefit of the higher utility to the user of the system is demonstrated in the next experiment.

-	laptop	office	home	desktop	ec2	gogrid	phone
laptop	-	0.5	3	3	1.5	2	0.3
office	0.5	-	1	1.1	2	3	0.3
home	2.1	1	-	3	2	1.6	0.3
desktop	3	1.4	3	-	1	3	0.2
ec2	1.6	3	2	1	-	2	0.3
gogrid	2	3	1.7	3	1.9	-	0.3
phone	0.3	0.3	0.2	0.3	0.3	0.3	-

TABLE II
Utility Matrix for the experiment of Figure 6

2) Multi-hop transfer scenario: In this experiment we show the results of a transfer simulation showcasing the effect of multi-hop transfer schedules aimed at high bandwidth utilization generated by Dexferizer. We use the device ensemble and topology shown in Figure 4.

We employ a synthetic workload for this experiment. The total number of items to be replicated is 155 and the sizes of the items vary between 4 and 25 MB. Initially, there are either 1 or 2 copies of each item, and the locations are chosen randomly. We create 1 new replica for each item, and the target location is also selected randomly. The optimizer routing algorithm is configured to favor the higher bandwidth paths whenever possible. Cells of the utility matrix are set to bandwidth values generated by the routing algorithm. Figure 7 illustrates the comparison of two approaches.

Transfer durations for many of the items get smaller with the optimized schedule. Transfer schedules generated by Dexferizer employ BitTorrent style downloads whenever possible and the routes generated are aimed at using the highest bandwidth paths between nodes. The naive approach always performs the transfer from the first node it finds having the item, and by default always uses the one-hop links between source and destination devices.

B. Tuning the optimizer with policies

The results we present in this section showcase the tunability of Dexferizer using user-specified transfer policies.

1) A transfer policy in action: In this experiment, we insert a transfer policy that is mentioned in section IV into Dexferizer. The policy says “Do not use a phone as a transfer source to replicate items”, and looks like:

```
xfer_policy([any_item], [phone_device], [any_device],
            [xfernone], [any_link]).
```

For this experiment, we use a generated workload. Each item in the data set has up to 3 initial copies that are randomly distributed over all the devices in the ensemble. Up to 3 new replicas of each item are going to be created on the rest of the devices (also randomly chosen). Figure 8 illustrates the effect of the policy on the generated transfer schedule. Activation of the policy dramatically reduces the amount of data traffic originating from the phone. Without the policy, Dexferizer maximizes the achievable bandwidth without considering the cost or energy constraint. Note that the number of times the phone is involved is still not 0 even when the policy is active. This is because some of the data items had only one copy at the time the request was issued and that single copy was located on the phone.

2) A priority policy in action: In this experiment we observe the effect of inserting item prioritization policies into Dexferizer. We compare the two different transfer schedules generated by the optimizer. The prioritization policy we insert into the optimizer is the following Prolog fact:

```
xfer_priority([[picture_item, private_item], [video_item]]).
```

which says that the photo items that are tagged as private

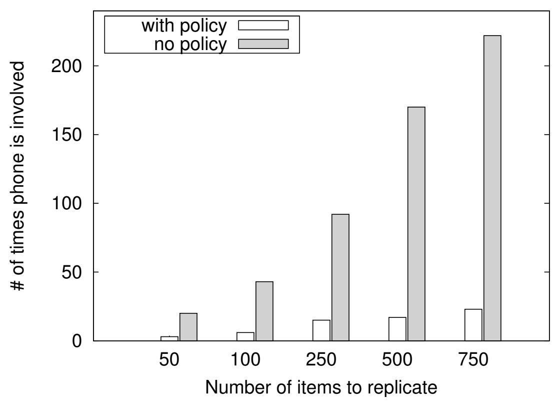

Fig. 8. Transfer policy in action.

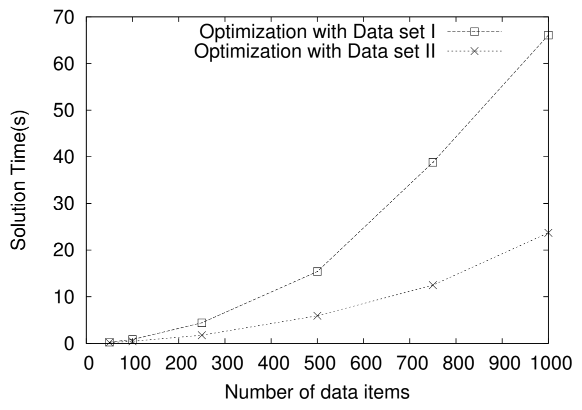

Fig. 10. Performance figure for the vanilla approach.

by the user should be treated with the highest priority in a transfer situation. The video items are of the second highest priority and the transfer of the rest of the items in the system will follow after these two. Figure 9 shows two CDFs that belong to transfers of all the photo items. The data set in this experiment has 1000 data items consisting of photos, mp3s, and video items. One of the CDFs belongs to the transfer schedule generated when there is no active priority policy in the system and the other one is the CDF in the presence of the policy.

The figure shows that the set of private photos (about half of the total photo collection) get scheduled earlier. As also depicted in the figure, the CDF with the priority policy stays constant for a certain period. This is due to the fact that the video items are scheduled for that particular period of time, as this was specified by the policy.

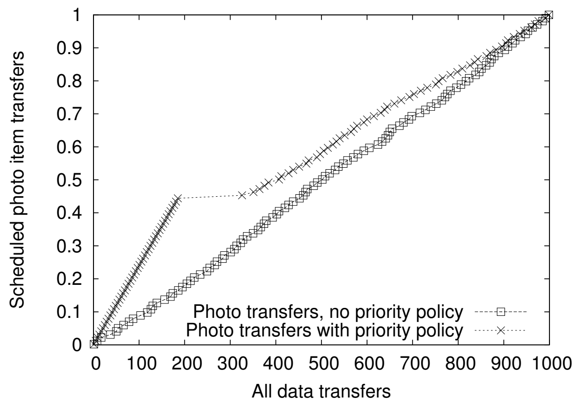

Fig. 9. Priority policy in action.

C. Optimizer scalability and resource usage

The next set of experiments are aimed to observe the limitations and resource consumption of our current implementation of Dexferizer. In these experiments, the CLP solver was running on a laptop that has a 2.5 GHz Intel Core 2 Duo processor, and 2 GB of memory.

1) Number of requests vs. solution time: Figure 10 shows the scaling of the basic CLP optimization with regard to number of items that need to be replicated. The data set labeled with I in this experiment is the same data set that is used in the experiment of Figure 8. Data set II is generated in a similar way. The difference in data set II is that the numbers of initial copies and new replicas were both set to 2. As depicted by the figure, the numbers of initial copies and new replicas changes the search space significantly, and therefore largely influences the solving time. We envision that, in most of the ordinary usage scenarios (except for disaster scenarios where several thousands of items might need to be moved around at the same time), the optimizer does not have to scale up to very many items. In practice, most of the time the system needs to deal with a few items at a time, e.g., after taking 10 photos. If scalability does become a problem, we then plan to address the issue by grouping the items into equivalence classes which can be derived by looking at the active item predicates that exist in the system.

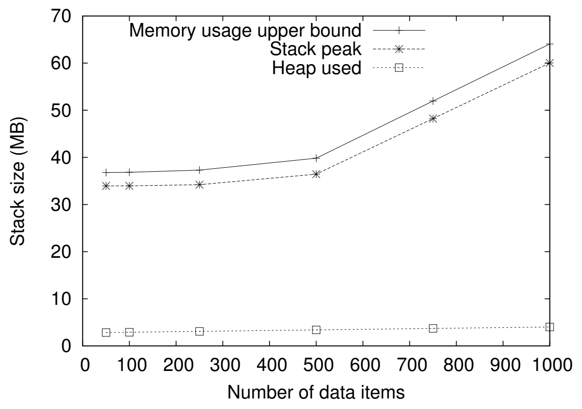

Fig. 11. CLP solver memory consumption.

2) Memory consumption: One last aspect we examine about the solver in the context of this paper is the resource consumption of the CLP solver. Figure 11 illustrates the solver's upper-bound memory consumption. By upper-bound memory consumption, we mean the total heap space and the sum of four different stack peaks (storing Prolog variables, backtracking information, checkpoints, etc). The peak value gives the maximum allocated during a session. As the figure shows, the memory consumption of the optimizer seems to scale better as compared to the solving time for the same data set.

VII. Conclusion and Future Work

In this paper, we presented an approach to optimizing the transfer of data objects within a user's collection of devices. We demonstrated that our approach can accommodate flexible user specified transfer policies that are not easily implemented with existing solutions. Moreover, we conclude that our approach shows significant potential for savings in time, power, bandwidth and other costs.

In terms of usability, Dexferizer is intended for both the users and developers, with being more geared towards the latter at the moment. For improving usability of the system from an end-user's perspective, we are planning to develop a graphical user interface (GUI) that can help users specify their transfer policies in a more intuitive way. This GUI then can translate user's policies into pure Prolog facts. We are also interested in finding out potential set of commonly used libraries of transfer policies that can be served to the users as off-the-shelf components.

As part of the future work, we are considering various extensions to our framework. We think that our declarative optimization framework can incorporate more information that can help improve the quality of transfer optimization even further. For example, each paid service such as a 3G connection or a rented cloud virtual machine has its own pricing model. We would like to be able to integrate various pricing models into declarative framework and reason about this data in an intelligent way.

We also think that in order to further optimize and support features such as deadlines on transfer requests, we can include personal usage profiles and daily connectivity information into the optimization process. This can be done with a three-dimensional matrix having the list of devices on the x-axis, one complete day on a 1-hour granularity (hence the 24 rows) on the y-axis, and the type of different links the devices can have on the z-axis. We think that this feature can enhance the quality of optimization and late-binding we can provide.

As a longer term future work, we are planning to explore the problem in the context of data centers and larger networks. We want to examine how our formulation of the problem fits with the conditions of data center networks which have different networking characteristics than personal data replication systems.

References

[1] V. Ramasubramanian, T. L. Rodeheffer, D. B. Terry, M. Walraed-Sullivan, T. Wobber, C. C. Marshall, and A. Vahdat, "Cimbiosys: a platform for content-based partial replication," in *Proceedings of the 6th USENIX symposium on Networked systems design and implementation (NSDI '09)*, 2009, pp. 261–276.

[2] B. Salmon, S. W. Schlosser, L. F. Cranor, and G. R. Ganger, "Perspective: semantic data management for the home," in *Proceedings of the 7th conference on File and storage technologies (FAST '09)*, 2009, pp. 167–182.

[3] N. Azzouna and F. Guillemin, "Analysis of ADSL traffic on an IP backbone link," in *Proceedings of the 2003 IEEE Global Telecommunications Conference. (GLOBECOM '03)*, December 2003.

[4] N. Tolia, M. Kaminsky, D. G. Andersen, and S. Patil, "An architecture for Internet data transfer," in *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI '06)*, May 2006.

[5] J. Su, J. Scott, P. Hui, J. Crowcroft, E. De Lara, C. Diot, A. Goel, M. H. Lim, and E. Upton, "Haggle: seamless networking for mobile applications," in *Proceedings of the 9th international conference on Ubiquitous computing (UbiComp '07)*. Springer-Verlag, 2007, pp. 391–408.

[6] A. Qureshi and J. Guttag, "Horde: separating network striping policy from mechanism," in *Proceedings of the 3rd international conference on Mobile systems, applications, and services (MobiSys'05)*. ACM, 2005, pp. 121–134.

[7] H. J. Wang, R. H. Katz, and J. Giese, "Policy-enabled handoffs across heterogeneous wireless networks," in *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications (WMCSA '99)*. IEEE Computer Society, 1999, pp. 51–60.

[8] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system," in *17th ACM Symposium on Operating Systems Principles (SOSP '99)*, Charleston, SC, December 1999.

[9] B. Ford, J. Strauss, C. Lesniewski-Laas, S. Rhea, F. Kaashoek, and R. Morris, "Persistent personal names for globally connected mobile devices," in *Proceedings of the 7th symposium on Operating Systems Design and Implementation (OSDI '06)*. USENIX Association, 2006, pp. 233–248.

[10] R. R. Kompella, A. Greenberg, J. Rexford, A. C. Snoeren, and J. Yates, "Cross-layer Visibility as a Service," in *Proceedings of 4th Workshop on Hot Topics in Networks (HotNets IV)*, 2005.

[11] E. A. Brewer, Y. H. Katz, Y. Chawathe, S. D. Gribble, T. Hodes, G. Nguyen, M. Stemm, and T. Henderson, "A network architecture for heterogeneous mobile computing," *IEEE Personal Communications*, vol. 5, pp. 8–24, 1998.

[12] J. Crowcroft, S. Hand, T. Roscoe, R. Mortier, and A. Warfield, "Plutarch: An argument for network pluralism," in *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA '03)*, 2003, pp. 258–266.

[13] J. T. Wroclawski, "The Metanet. White Paper," in *Proceedings of Workshop on Research Directions for the Next Generation Internet*, 1997, White Paper.

[14] S. R. Das, C. E. Perkins, and E. M. Belding-Royer, "Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks," in *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '00).*, March 2000, pp. 3–12.

[15] C. E. Perkins and E. M. Royer, "Ad-Hoc On-Demand Distance Vector Routing (AODV)," in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, February 1999, pp. 90–100.

[16] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," *Internet RFCs*, vol. RFC 3626, 2003.

[17] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '03)*. New York, NY, USA: ACM, 2003, pp. 27–34.

[18] "Swisscom tariffs for mobile internet," http://www.swisscom.ch/res/mobile/internet/index.htm?languageId=en&campID=md.

[19] "Swisscom standard tariff for data transmission abroad." http://www.swisscom.ch/res/mobile/international/data/index.htm.

[20] "Dropbox: Online backup, file synchronization and sharing software." http://www.dropbox.com.

[21] "Nokia N900 Hardware Power Consumption." http://wiki.maemo.org/N900_Hardware_Power_Consumption.

[22] "The Anzere personal storage system." http://www.systems.ethz.ch/research/projects/anzere.