HomeViews

Geambasu et al., 2007

Presented by Jeremy Handcock
Motivation

• We have a lot of personal digital data

• Existing solutions for organizing and sharing aren’t ideal
Typical Scenario
Existing Solutions: Sharing

• One must manually upload content to centralized repository
• Centralized repositories typically require registration for new users
• Inevitably you end up with multiple copies
• Dynamic data isn’t supported
Existing Solutions: Organizing

- Desktop search tools
- Create abstract data views or smart folders
- Requires metadata for non-textual objects
Existing Solutions: Organizing
Idea!

- Decentralize data using P2P
- Share data using location-independent abstract views
- Provide lightweight access control
- Put it all in a middleware framework
Query Processing

Query

Query Executor

Local?

Yes
- Run Local Query

No
- Contact Peer to Run Query
Query Processing

Query

Local Query Engine

Local File System

{ Beagle, Spotlight, ...

...
Capabilities

- Associated with a view
- Self-authenticating unit of permission
- Capabilities unforgeable with high probability
- Used in global naming and routing
# Capabilities

<table>
<thead>
<tr>
<th>global view ID</th>
<th>password</th>
<th>IP hint</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

| Hash of node’s MAC + Locally-unique view ID | 128-bit random password |
Capabilities

• Capability provides location information

• IP hint hopefully points to node where capability is defined

• When IP Hint fails, use a DHT (Chord) to locate the content
Capabilities

- Capability could be represented as a simple “token”
- Distributing capabilities is annoying, but relatively easy
- Revocation is easy
Capabilities

- Is this an appropriate model for sharing personal digital data?
- Do capabilities provide sufficient security?
Queries

• Extension of SQL
• Fundamental difference: relations replaced with capabilities
• Define a capability by creating a view
• Base capability provides access to all files
CREATE VIEW MyFiles2007 AS
SELECT * FROM BaseCapability
WHERE createDate > '2007-01-01'
AND createDate < '2008-01-01'

New capability: Jeremy2007
Queries: Restrictions

RESTRICT Jeremy2007
RIGHTS SELECT

→ New capability: Jeremy2007_RO
Queries: Selections

SELECT * FROM Jeremy2007_RO
WHERE fileType = 'image'
CREATE VIEW MyFriendMovies2007 AS

SELECT * FROM Jeremy2007_RO
WHERE fileType = 'movie'
UNION
SELECT * FROM Rob2007_RO
WHERE fileType = 'movie'

New capability: FriendMovies2007
Evaluation: Applications

- Simple read-only filesystem
- Photo gallery
Evaluation: Applications
5.1 Evaluation of Simple Queries

In our system, views can be composed and distributed in various ways, either by applying a selection on top of another view or by growing a view that is defined directly over a base view. Complex queries may also grow, for example, by applying union or set difference to views. We now analyze the performance of more complex queries.

Simple queries allow us to identify and reason about the different components of our system on total query execution times. To evaluate simple queries, we measure the impact of different file sizes. For example, to experiment with a different file size, we initially define a view on the ID tags of a given depth. We then create a view of a given depth and a view that selected a theme with a query of size hgg. We created hgg files with the album IDj tags of di.

We controlled the query result size by appropriately setting attributes. Enable rich queries that are supported by yeagle. Our environment had no significant impact on our results. Measurements show that the hardware used was a 2.4GHz Pentium 4 with 1GB of memory. From our remote views, times form the bulk of total query execution even for small file sizes. The local query processing time represents a major component of query execution.

Figure 7: Evaluation: Numbers

Figure 7 breaks down query execution time into components. The chart shows the evaluation time on slow networks for different file sizes. Execution is fast for medium-size results both for local and remote views. Query execution times for larger results are noticeable over slow connections. Table 7 shows HomeViews overhead increases slowly with result size. Caching of local query results is negligible, although the query parsing, view definition lookup in the local catalog, and zapability validation time and other HomeViews overhead are considered.

Query time becomes noticeable over slow connections. For local and remote configurations, we also vary the query result size. Our queries return file names. Simple queries are one level deep and involve views whose definitions include multiple other views. Complex queries are one level deeper and involve a single view.

Table 4: Other

<table>
<thead>
<tr>
<th>Size</th>
<th>Local</th>
<th>LAN</th>
<th>Broadband</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>500</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

For local and remote configurations, expected query execution times if Spotlight were used instead of Beagle query time are computed from Table 3. Beagle query time is about 10 times faster than Spotlight. The other HomeViews components, which makes this a good approach.

In our system, views can be composed and distributed in various ways. For example, to experiment with a different file size, we initially define a view on the ID tags of a given depth. We then create a view of a given depth and a view that selected a theme with a query of size hgg. We created hgg files with the album IDj tags of di.

We controlled the query result size by appropriately setting attributes. Enable rich queries that are supported by yeagle. Our environment had no significant impact on our results. Measurements show that the hardware used was a 2.4GHz Pentium 4 with 1GB of memory. From our remote views, times form the bulk of total query execution even for small file sizes. The local query processing time represents a major component of query execution.

Figure 7: Evaluation: Numbers

Figure 7 breaks down query execution time into components. The chart shows the evaluation time on slow networks for different file sizes. Execution is fast for medium-size results both for local and remote views. Query execution times for larger results are noticeable over slow connections. Table 7 shows HomeViews overhead increases slowly with result size. Caching of local query results is negligible, although the query parsing, view definition lookup in the local catalog, and zapability validation time and other HomeViews overhead are considered.

Query time becomes noticeable over slow connections. For local and remote configurations, we also vary the query result size. Our queries return file names. Simple queries are one level deep and involve views whose definitions include multiple other views. Complex queries are one level deeper and involve a single view.

Table 4: Other

<table>
<thead>
<tr>
<th>Size</th>
<th>Local</th>
<th>LAN</th>
<th>Broadband</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>500</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

For local and remote configurations, expected query execution times if Spotlight were used instead of Beagle query time are computed from Table 3. Beagle query time is about 10 times faster than Spotlight. The other HomeViews components, which makes this a good approach.
The main goal of these systems is to facilitate the sharing and composition of views in a PfP environment to organize personal data. Our work builds on the same idea of using views to organize personal information. In particular, the Haystack implementation has begun to explore new techniques for organizing and searching data. Users can create database-style views over their data. In HomeViews, applications can remotely share data, but our measurements demonstrate the negligible cost of our protection mechanism and the practicality of our approach for medium-scale environments. Peer-to-peer systems have become popular for sharing digital contents. In recent years, tools such as Spotlight and Mac OS X have been proposed to allow registered users of an administrative domain to access resources from another administrative domain without requiring registration with the perspective of sharing personal information, however, these systems thus focus on powerful and efficient search and replication on system performance is beyond the scope of our current study.

Sharing and protection are accomplished without central coordination of any kind. Pose views, and seamlessly integrate local and remote views. With HomeViews, applications can easily create views, compute integrated view definitions and share them. We implemented two applications on top of HomeViews: a simple file-sharing protocol and a port of the Gallery photo-sharing application. Our experience with Gallery in particular shows that capabilities are readily applied to operating systems and databases to enable access control that can be encrypted once and published. More dynamic selective sharing techniques are to encrypt data with multiple keys and distribute different keys to different users. HomeViews has no such requirement. It integrates concepts and mechanisms from capability systems, and seamlessly integrates local and remote views.

The capability protection model has been previously applied to operating systems and languages, and architectures. It is a practical model for medium-scale environments. Peer-to-peer middleware systems that simplify the construction of distributed personal information-sharing applications. HomeViews facilitates ad hoc, peer-to-peer sharing of data. We prototyped HomeViews in a Linux environment using a port of the Gallery photo-sharing application and replication on system performance is beyond the scope of our current study. Peer-to-peer systems have become popular for sharing digital contents. In recent years, tools such as Spotlight and Mac OS X have been proposed to allow registered users of an administrative domain to access resources from another administrative domain without requiring registration with the perspective of sharing personal information, however, these systems thus focus on powerful and efficient search and replication on system performance is beyond the scope of our current study.

Our microbenchmarks show the parameters that characterize our system’s performance and enable us to derive the efficiency of these mechanisms. The capability protection model has been previously applied to operating systems and languages, and architectures. It is a practical model for medium-scale environments. Peer-to-peer middleware systems that simplify the construction of distributed personal information-sharing applications. HomeViews facilitates ad hoc, peer-to-peer sharing of data. We prototyped HomeViews in a Linux environment using a port of the Gallery photo-sharing application.

**Evaluation: Numbers**

![Bar chart showing the relationship between result size and throughput](image)

- **Completed requests/sec**: The chart illustrates the throughput in terms of completed requests per second for different result sizes.
- **Result size**: The chart plots the result size on the x-axis.
- **Throughput**: The y-axis represents the completed requests per second.
- **Data points**: The graph shows a significant drop in throughput as the result size increases, indicating diminishing returns for larger result sizes.

The chart visually demonstrates that HomeViews is efficient in handling smaller result sizes, with a sharp decline in performance as the result size grows larger. This suggests that optimizing query evaluation for smaller result sizes could significantly improve performance and scalability, especially in medium-scale environments.
Applicability to OSNs

• Middleware platform for OSN applications to share data between users: ie. PeerSpective, Facebook

• Capability management could be hidden
Applicability to OSNs

- Danger: lack of explicit sharing action and dynamic data
Conclusions

• P2P not always appropriate model for sharing personal digital data
• Complex views offer opportunities for poor performance
• We still don’t have an ideal solution