Selective Replication for Content Management Environments

Content management applications typically depend on information stored in both relational database tables and operating system files. Often, content providers replicate all or parts of the available database data and associated files to increase application availability, address resource constraints and costs, or better support geographically dispersed and mobile users. This article presents a solution that addresses integrated and selective database and file replication in the context of an enterprise voice portal. The solution is transparent to existing applications and imposes minimal storage overhead.

Euthimios Panagos
Telcordia Applied Research

Alex Delis
University of Athens

Web applications often depend on content management systems for delivering a rich user experience. Typically, these systems store content created using desktop applications or captured via a combination of software and hardware (as with audio clips) in files. They also use database tables to store metadata that describes attributes, associations, classification hierarchies, and access restrictions associated with the content. Content providers tend to replicate such database data and associated files to better support geographically dispersed and mobile users, increase application availability, or cope with resource constraints and costs. In many cases, however, it's only necessary to replicate a portion of the available information. We don't generally need to replicate highly localized content or content that doesn't correspond to business- or mission-critical assets.

To date, no out-of-the-box solution exists for selectively replicating database table entries and their associated files. Although various techniques such as a combination of database snapshot replication[1,2] and file-system replication are available, such solutions aren't integrated and, typically, depend on coupled application, database, and file-system designs. The problem becomes more challenging when we need to retrofit replication into existing systems without making application changes.

Much recent work in content delivery networks (CDN) addresses replication.[3–5] Content providers usually replicate Web pages and images among Web or application servers to improve response times in CDN environments. However, this work doesn't address integrated database and file-system replication.

This article presents a method for supporting integrated and selective replication between database tables and files in an enterprise voice portal.[3] Our solution imposes minimal storage overhead and is transparent to existing applications. In addition, it supports dynamic changes to the information we need to replicate and the voice portal sites that participate in the replication.

Voice Portal

A voice portal is a Web service that users can reach by telephone for information such as stock quotes and prerecorded audio content. It unites the Internet and telephone network via speech and touchtone interfaces. In this article, we offer a high-level overview of the specific voice portal in which we implemented our selective and integrated replication solution. More detailed information about the portal is available elsewhere.[3]

Information Model

The portal's fundamental unit of content is a *story*, which consists of media components (called *story items*) and pertinent metadata. Story items are multimedia files, which can exist in several formats to support different access devices (telephone and Web browser, for example). Stories can also reference other stories via unique story IDs.

Story metadata is textual and consists of fixed attributes (such as title, time stamp, and length) and name-value keyword pairs. Content administrators define the categories that can be used as keyword names (such as `company name`) and select values for them from enumerated or unrestricted sets. Speech-recognition applications handle enumerated sets well because they translate easily into finite vocabulary lists. Speech recognizers that don't support natural-language recognition can't handle unrestricted sets, although such sets are compatible with Web access.

Content administrators manually group stories into *channels*, which represent thematic classification structures for grouping stories that are related in some way. (The Executive Commentary channel might contain interviews with company executives, for example.) A story is always created in the context of a specific channel, but story authors and administrators can later associate it with other channels, subject to access restrictions, via a Web interface.

Content administrators organize channels into one or more category hierarchies to facilitate the navigation and location of stories for both Web and telephone user interfaces. Individual users can personalize these hierarchies through a Web interface to better suit their preferences.

Providers, Users, and Access Control

The voice portal recognizes two service entities: providers and users. Providers represent the business rules associated with information capture and delivery. Among other things, they are responsible for determining the

- hierarchy of categories and channels in which stories are created and stored,
- keyword categories and enumerations available during story publishing and editing, and
- user privileges for accessing (parts of) the category and channel hierarchy.

Users are associated with specific providers, and they can access voice portal services only after they successfully authenticate using numerical IDs and personal identification numbers (PINs) — speech recognition doesn't reliably handle alphanumeric IDs, which are also harder to handle from numeric keypads. Channels and their stories are subject to access control based on closed user groups comprising users, channels, and access rights. When users access channels, their access rights are the union of the rights of all closed user groups containing the same channel and user.

Overall Site Architecture

As Figure 1 shows, a user can access a voice portal site from the Internet using a Web browser or from a telephone network using any landline or cellular telephone. Telephony gateways offer content management functionality using speech recognition or touchtone and dial-out capabilities for content delivery in context broadcasts or user-specified alerts — for example, call me when a story appears in my Executive Commentary channel.

The voice portal stores information about providers, users, stories, keywords, and so forth in a relational database. Media files are stored in a file system (media store). A Java2 Enterprise Edition (J2EE)-based middleware layer (part of the application server) offers content management functionality, while a client-side presentation layer uses JavaBeans to render the appropriate user interfaces (such as HTML and VoiceXML).

Streaming servers stream live audio and video content over the Web and telephone. A notification server implements custom alerts and content broadcasts. This server is responsible for alerting users, via a telephone call or email, when new content that matches their interests is available. In addition, this server carries out content broadcasts to distribution lists containing tens to thousands of recipients.

Physical Storage Organization

The voice portal stores media files in *volumes*, which correspond to file-system directories that reside in redundant arrays of inexpensive disks (RAIDs). All related story items are stored under one story directory, located under a volume directory. The volume directory structure is specified by expressions that include date information — for example, `yyyy/mm/dd` denotes a hierarchy in which year, month, and day are the first, second, and last subdirectories, respectively. Under this directory structure, all items for a story created on 4 April 2002 are stored under `/mdir/vdir/2002/04/04/sdir`, where `mdir` is the mounting point of the volume directory `vdir` for a specific machine, and `sdir` is the story directory.

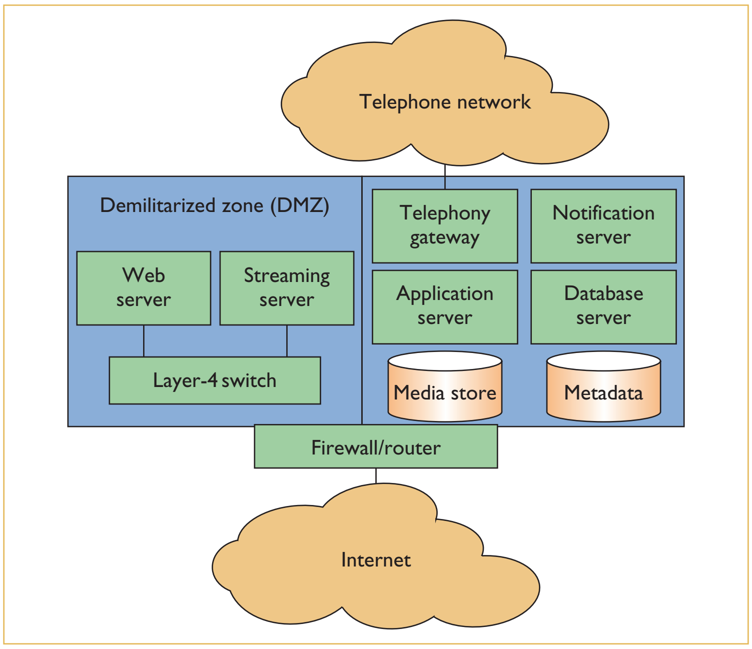

Figure 1. Physical architecture of a voice portal site. Content is stored in a file system, whereas metadata is stored in a relational database. The Web and voice interfaces are offered by Web servers and telephony gateways, respectively.

Database and File-System Replication

Database and file-system replication is not new. However, researchers have not adequately addressed integrated replication between specific database table entries and the operating system files referenced by these table entries so far. We faced several challenges when implementing selective and integrated replication in our voice portal.

Selective Replication

When several financial institutions started using the voice portal services in early 2001, selective replication of content and user accounts between sites became a requirement. These organizations have offices around the globe, and some content created in one location is of interest to others. Also, some users are frequent travelers and require access to their profiles (including all available and accessible profile stories) from anywhere.

In addition to client-driven requirements, an automated telephone broadcast service, which was made available as part of the voice portal services in late 2002, also required selective replication. This service supports automated telephone broadcasts of stories to distribution lists associated with the specific channels in which these stories were created. Distribution lists can contain thousands of telephone numbers from all over the world.

Because each voice portal site has a fixed capacity in terms of the number of concurrent telephone calls it can handle, multiple sites might have to participate in story broadcasts to large numbers of telephones. (The automated telephone broadcast service might have to place calls to all distribution list members concurrently — as with Wall Street firms that use this service regularly.) To avoid poor audio quality due to delays in fetching relevant content from remote sites, the broadcast stories must be replicated to all participating sites.

Replication Challenges

The initial implementation of voice portal applications did not offer any provisions for replication, such as globally unique IDs. To make our replication scheme transparent to existing applications, we thus had to address several problems:

- Many system entities (such as channels and volumes) are assigned numerical keys that aren't globally unique. To preserve uniqueness, we must generate new primary keys for replicated database entries.
- To facilitate direct access to story items, story

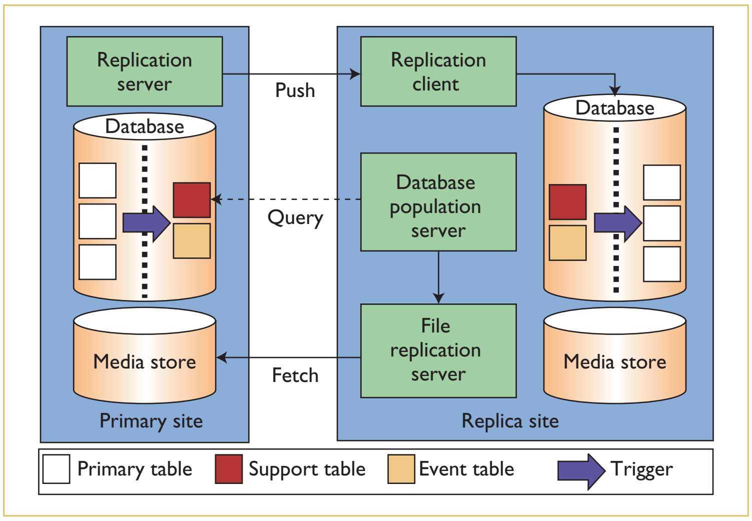

Figure 2. Replication architecture. Database triggers capture updates at primary sites. These updates are pushed to replica sites by the replication server, where a replication client records them in the local database. A database population server installs all updates to the replica site and coordinates with the file-replication server to fetch appropriate content files.

IDs include the IDs of the volumes that contain their items. Therefore, we must rewrite story IDs when replicating them to volumes with different IDs.

- Stories can reference each other via story IDs. Because referenced stories might be associated with channels that aren't replicated, we must manage story references at each replica site.

In addition, our solution must support content administrators' replication policies. Depending on the stored content's importance, for example, administrators might dynamically select which channels, stories, and user accounts to replicate to different sites.

Furthermore, our solution must support the automated broadcast service's requirements. Namely, it must be able to select which sites will be part of a story broadcast based on the following criteria:

- *Site capacity*. Given that each site has a fixed number of telephone lines (T1/E1 lines) available, multiple sites might have to carry out individual broadcasts, depending on site capacities and distribution list sizes.
- *Telecommunications costs*. Because the cost of making a phone call depends on the call's origin, multiple sites might again have to carry out a broadcast to reduce costs when calling distribution list members located around the globe.

Finally, our broadcast site selection and story-replication solution must be dynamic to let service providers create and delete distribution lists and initiate story broadcasts at any time.

Replication Architecture

As Figure 2 illustrates, our replication architecture combines asynchronous, store-and-forward database replication[1,2,6–8] with remote file access (via the Network File System [NFS] or Server Message Block [SMB] file-sharing protocol). Custom database triggers record, in a persistent queue, all updates to database entries marked for replication in a primary site. These updates are propagated asynchronously to replica sites, which fetch created and updated content files and store them locally.

Our solution detects changes to database table entries and captures these changes by using SQL `insert` and `update` triggers and several support tables. We install triggers on all tables that might contain rows that require replication (such as provider, story, and channel). Support tables store replication configuration parameters, such as replica site information and IDs for replicated channels and accounts.

When a trigger fires, it queries the support tables to determine whether to capture the update. For example, when a voice portal user creates a story in a channel, the fired trigger queries the table containing channel replication information to determine whether to capture the story creation event. If it locates an entry for the channel, it captures the update. Otherwise, the trigger queries the table containing provider replication information; if it finds an entry for the provider that owns the channel, it captures the update. This approach supports dynamic changes to replication configuration.

After capturing an update, the trigger inserts information about it into an outgoing event table. This information includes the updated entity's type (for example, story item), its primary key, the operation's time stamp, and a status field indicating whether the entity is new. Here is the logic we use for inserting entries into the event table:

```
IF operation is creation THEN
  INSERT INTO event_table
  VALUES (type,key,timestamp,
        'new');
  ELSE IF event_table contains key THEN
    UPDATE event_table
    SET timestamp=value WHERE key=value
  ELSE
    INSERT INTO event_table
    VALUES (type,key,timestamp,
        'old');
  ENDIF
```

The most important feature of the event table insertion logic is that no duplicate entries exist for any given entity. Hence, the database space required for capturing replication information is bounded and doesn't depend on how often database entries are updated.

Update Propagation

At a primary site, the *replication server* maintains a replication status table through which it tracks replication progress for entries in each replica site's event table. The replication server scans the event table periodically (based on a configuration option) and retrieves all entries with greater time stamps than the maximum in the event table entries selected during the previous scan. Then, it issues one or more queries to retrieve the actual information for each updated or created entry found during the scan, encodes the information in XML messages, creates entries in the replication status table, and inserts each XML message into a memory queue. Worker threads retrieve these messages from the queue and push them to replica sites using secure HTTP.

At a replica site, the *replication client* receives the updates, stores them in the appropriate incoming event tables (there's one for each entity type that's replicated), and acknowledges their receipt. After receiving all acknowledgements, the replication server deletes the appropriate entries from the replication status table and updates their corresponding entries in the event table – it deletes entries for which time stamps weren't updated and sets the `status` field to "old" for entries with updated time stamps.

At a replica site, *database population server* installs the updates in the local database by using the entries in the incoming event tables and the entries in the primary key-mapping tables (see the next section for more details). The database population server performs these updates in a specific order to avoid foreign key constraint violations. Once the updates are installed, it deletes all processed incoming event table entries.

The *file replication server* is responsible for replicating media files, which it fetches from the primary site over HTTP.[9,10] The database population server provides information about which media files to fetch using direct communication over sockets. To guard against crashes, the database tracks this process.

Avoiding Uniqueness Conflicts

Uniqueness conflicts are possible in the voice portal because each voice portal site enacts the same numerical key generation process. This process uses a database table, named `primary_keys`, which contains a row for each table that requires primary keys. Each such row contains the name of a table and an integer value. When a new primary key needs to be generated for a table, the key-generation process locates the entry for this table in `primary_keys` and returns the integer value after incrementing it by one in the context of a transaction.

To avoid uniqueness conflicts, the database population server generates new primary keys for replicated entries. The mappings between primary key values (at the primary and replica) are maintained in key-mapping tables that track volume, channel, account, profile, and category IDs. Provider IDs are unique across all sites, and login IDs are unique for each provider. Therefore, it's unnecessary to map them.

Story IDs comprise three parts: the ID for the volume in which the story items are physically stored, the story's creation time stamp, and a counter. The counter, a volume property, ensures unique IDs for stories created on the same date and volume. Therefore, the volume ID mapping table suffices both for mapping story IDs between primary and replica sites and rewriting story IDs at replica sites.

Story item names correspond to the names of the files where they are physically stored (for example, `body.wav`). These names are unique for a given story. A story item's name and ID thus guarantee an item's uniqueness, and so we use them as primary keys.

Maintaining Story References

When a story containing story references is replicated, the references must either be set to `null` or point to local copies of stories. We opted for the latter whenever possible. Here, we had to handle two cases:

- The referenced story is associated with at least one channel that's explicitly or implicitly replicated.
- The referenced story is associated with channels that aren't replicated.

A channel is explicitly replicated when an entry for it exists in the channel-replication table. A channel is implicitly replicated when an entry for the provider that owns the channel exists in the provider replication table.

In either case, we initially set references to `null` and record the story association into a story-reference table. A background thread in the database population server scans this table periodically and queries the primary site (or sites) for all unprocessed entries to determine which case applies to each referenced story. In the first case, the thread marks the corresponding table entries in this table as "local," whereas the entries are marked as "remote" in the second case.

A local entry implies that the referenced story is either already replicated or will be soon. The thread periodically checks whether the referenced stories have been replicated. To do so, the thread rewrites the story IDs, using the volume ID mapping table, and queries the story table. When this thread locates a local copy of the referenced story, it updates all stories referencing it and drops the old entry from the story reference table.

A remote entry implies that the referenced story won't be replicated under the current replication configuration, but it might be replicated in the future. We can either delete such entries from the story reference table and use `null` references, or keep the entries and have the background thread include them when querying the primary sites until their states change to local, the stories referencing them are deleted, or the associations are broken. As a policy, we selected the former option.

Dynamic Site Selection

When a voice portal user creates a new story broadcast, we must use broadcast size and delivery costs to compute which sites should participate in the story's delivery. Broadcast size determines the required capacity (in terms of telephone lines), and delivery costs correspond to the charges associated with all calls that must be made as part of the broadcast. The goal is to minimize both the number of participating sites and the delivery costs. We assume, of course, that the required broadcast capacity is less than the available capacity across all sites.

We developed an approximation algorithm for selecting the sites that will participate in the story's delivery. Our algorithm is biased toward keeping delivery costs low by minimizing the telephone charges associated with expensive deliveries by assigning such deliveries to voice sites with low per-minute call costs. The main logic is as follows:

1. Group broadcast recipients based on the country codes in their telephone numbers. Let $D = \{D_1, D_2, \ldots D_N\}$ be the set containing the groups of recipients whose telephone numbers contain the same country code.
2. For each D_i in D, compute the average telephone charges associated with story deliveries to all recipients of D_i. Compute average charges as $|D_i| * SL * CD_i$, where SL is the story length in minutes and CD_i is the average per-minute cost of a phone call to a recipient in D_i across all voice sites. Let $D' = \{D'_1, D'_2, \ldots D'_N\}$ be a nonincreasing ordering of D based on the computed average telephone charges.
3. Iterate over all entries in D', and for each D'_i, assign its recipients to voice sites with available capacity and the lowest per-minute costs for calls to those recipients. A 2D matrix, C, where cell C_{ij} contains the per-minute cost of making a call from voice site S_i to a telephone in country D_j, facilitates this assignment.

The replication server (see Figure 2) executes the site-selection algorithm in response to a story creation's capture in a channel associated with a broadcast distribution list. When a content administrator or authorized user creates a broadcast distribution list, the voice portal associates this list with a channel, which is automatically inserted into the table containing channel-replication information. Once we've identified the sites that will participate in the broadcast, our solution replicates the story and relevant parts of the distribution list to them.

Implementation Discussion

In the past, poorly implemented database triggers were considered to be performance bottlenecks. However, modern database management systems offer efficient trigger implementations by compiling trigger code and, in many cases, letting database administrators pin the compiled trigger code in memory.

Our selective replication implementation keeps trigger-processing overhead even lower by requiring only a small number of triggers and minimizing the number of table lookups performed within each. This might not be the case for applications that follow our approach but require considerable processing within each trigger.

The ability to identify which files need to be replicated from database content is an important aspect of our solution. The voice portal's specific physical storage architecture, coupled with its recording of the association between story items and media files in a database table, makes explicit monitoring of the file system unnecessary.

In the future, we plan to study how to extend our selective and integrated replication solution to support a wider range of applications. In particular, we plan to investigate how to integrate our solution with high-available and fault-tolerant J2EE application servers that implement custom Web applications using a combination of static and database-driven dynamic content.

Acknowledgments

This work was completed while Euthimios Panagos was at Voicemate.

References

1. *Oracle 9i Replication*, Oracle, white paper, June 2001; www.oracle.com/technology/products/dataint/pdf/oracle9i_replication_twp.pdf.
2. *Comparing Replication Technologies*, PeerDirect, white paper, 2002; www.peerdirect.com/Products/WhitePapers/.
3. Akamai Technologies, *Fast Internet Content Delivery with FreeFlow*, white paper, 1999; www.di.uoa.gr/~ad/MDE519.docs/Akamai-How.pdf.
4. N. Bartolini, E. Casalicchio, and S. Tucci, "A Walk through Content Delivery Networks," *Proc. IEEE/ACM Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecomm. Systems* (MASCOTS), IEEE CS Press, 2003, pp. 1–25.
5. J. Jung, B. Krishnamurthy, and M. Rabinovich, "Flash Crowds and Denial of Service Attacks: Characterizations and Implications for CDNs and Web Site," *Proc. Int'l World Wide Web Conf.* (WWW 2002), ACM Press, 2002, pp. 293–304.
6. A. Gorelink, Y. Wang, and M. Deppe, "Sybase Replication Server," *Proc. 1994 ACM SIGMOD Int'l Conf. Management of Data*, ACM Press, 1994, pp. 469.
7. B. Kemme, "Database Replication for Clusters of Workstations," PhD dissertation, Swiss Federal Inst. Technology, ETH no. 13864, 2000; www.cs.mcgill.ca/~kemme/papers/phd-dina4.pdf.
8. D. Scott, J. Krischer, and J. Rubin, *Disaster Recovery: Weighing Data Replication Alternatives*, Gartner research note T-13-6012, June 2001.
9. O. Kiselyov, "A Network File System over HTTP: Remote Access and Modification of Files and *Files*," *Proc. 1999 Usenix Ann. Tech. Conf.*, Usenix Assoc., 1999, pp. 75–80.
10. *WebNFS: Bringing a File System to the Internet, Sun Microsystems*, white paper; wwws.sun.com/software/webnfs/overview.html.

Euthimios (Thimios) Panagos is a senior research scientist at Telcordia Applied Research. His research interests are in distributed data management, event notification, workflow management, high-performance messaging platforms, and real-time communications. Panagos has a PhD in computer science from Boston University. He is a member of the ACM. Contact him at thimios@research.telcordia.com.

Alex Delis is a faculty member in the Department of Informatics and Telecommunications at the University of Athens. His research interests are in distributed data management, networked information systems, distributed computing, and computer system evaluation. Delis has a PhD in computer science from the University of Maryland, College Park. He is a member of the IEEE and the ACM, and has received an NSF Career Award and a Fulbright fellowship. Contact him at ad@di.uoa.gr.