Mobile Software Development
Framework: Mobile-Cloud Services (Push/Track)

10/18/2012

Y. Richard Yang
Outline

- Admin
- Mobile cloud service
  - Push notification service
  - Storage service
  - Track service
  - Split service
HW3 posted
Recap: Accessing Data in Cloud

- A typical design pattern is that a device updates/receives data in the cloud
  - Cloud as a rendezvous point

- Challenge: How do you keep data on a device fresh?
Recap: Solution Space

- Mobile poll

- Cloud push
  - Each app push
  - Shared (infrastructure) push
Shared Push Service

- A single **persistent** connection from device to a cloud push service provider
- Multiple application providers push to the service provider
- Service provider pushes to a device using the persistent connection

- Two examples
  - Apple Push Notification Service (APNS)
  - Google Cloud Messaging (GCM)
Design Requirements of a Shared Push Service

- Security/Authorization
  - Do not allow arbitrary app to push to a device

- Scalability
  - A large scale system may have millions of clients

- Fault tolerance
  - Client/device, push servers all can fail

- Generality
  - Can be used by diverse applications
Design Point: Authorization

Design 1: App does not know registered devices. Broadcast to all.

Design 2: App query registered devices; Multicast

Design 3: Device notifies registration ID to its server;
Design Point: What to Push?

- Option 1: Just push signal (data available) to devices and then devices fetch from app servers

- Option 2: push app data
Design Point: Reliability (What Can Go Wrong)

App sends to regIDs

RegID = Registration(DEV_ID, App_ID)

Device notifies regID to its server;
Soft State Design

- State at the third party is soft state if the entity who sets up the state does not refresh it, the state will be pruned at the 3rd party
Apple Push Notification Service

- iOS device maintains a persistent TCP connection to an Apple Push Notification Server (APNS)

A push notification from a provider to a client application.

Multi-providers to multiple devices
Device token Contains information that enables APNs to locate the device
- Client app needs to provide the token to its app provider
- Device token should be requested and passed to providers every time your application launches
Apple Push Notification Data

- Each push notification carries a payload
  - 256 bytes maximum
  - Best effort delivery

- App provider provides a JSON dictionary object, which contains another dictionary identified by the key `aps`

- `aps` specifies the following actions
  - An alert message to display to the user
  - A number to badge the application icon with
  - A sound to play
APNS Example: Client

1. - (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
2. {
3.   // Let the device know we want to receive push notifications
4.   [[UIApplication sharedApplication] registerForRemoteNotificationTypes:
5.   (UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeSound | UIRemoteNotificationTypeAlert)];
6.   return YES;
7. }

9. - (void)application:(UIApplication*)application
didReceiveRemoteNotification:(NSDictionary*)userInfo
10. {
11.   // userInfo contains the notification
12.   NSLog(@"Received notification: %@", userInfo);
13. }

13. - (void)application:(UIApplication*)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData*)deviceToken
14. {
15.   NSLog(@"My token is: %@", deviceToken);
16. }
APNS Example: Server

1. $devicetoken = 'f05571e4be60a4e11524d76e4366862128f430522fb470c46fc6810fffb07af7';
2. // Put your private key's passphrase here:
3. $passphrase = 'PushChat';
4. // Put your alert message here:
5. $message = 'CS434: my first push notification!';
6. $ctx = stream_context_create();
7. Stream_context_set_option($ctx, 'ssl', 'local_cert', 'ck.pem');
8. stream_context_set_option($ctx, 'ssl', 'passphrase', $passphrase);
9. // Open a connection to the APNS server
10. $fp = stream_socket_client('ssl://gateway.sandbox.push.apple.com:2195', $err, $errstr, 60, STREAM_CLIENT_CONNECT|STREAM_CLIENT_PERSISTENT, $ctx);
11. if (!$fp)
12.   exit("Failed to connect: $err $errstr" . PHP_EOL);
13. echo 'Connected to APNS'. PHP_EOL;
// Create the payload body
$body['aps'] = array(
    'alert' => $message,
    'sound' => 'default'
);

// Encode the payload as JSON
$payload = json_encode($body);

// Build the binary notification
$msg = chr(0) . pack('n', 32) . pack('H*', $deviceToken) . pack('n', strlen($payload)) . $payload;

// Send it to the server
$result = fwrite($fp, $msg, strlen($msg));

if (!$result)
    echo 'Message not delivered'. PHP_EOL;
else
    echo 'Message successfully delivered' . PHP_EOL;

// Close the connection to the server
fclose($fp);
Google Cloud Messaging

- Very similar to APNS

See http://developer.android.com/guide/google/gcm/gs.html for detailed steps
GCM Flow: App Developer Registration

- App developer registers a project at Google
  - Open API console: https://code.google.com/apis/console/

- After Create project

  https://code.google.com/apis/console/#project:4815162342

  Project ID; Sender ID
GCM Flow: Device App Registration

- Enable cloud to device messaging in your app
  - Add permissions in Manifest
  - App (on device) registers with Google to get registration ID

```java
public class MyActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        GCMRegistrar.checkDevice(this);
        GCMRegistrar.checkManifest(this);

        final String regId = GCMRegistrar.getRegistrationId(this);
        if (regId.equals("")) {
            GCMRegistrar.register(this, SENDER_ID);
        } else {
            Log.v(TAG, "Already registered");
        }
    }
}
```
Device App Handle Events

The GCMBroadcastReceiver (defined in GCM library) handles the broadcast messages, and calls methods defined in .GCMIntentService, if you define this service.

```xml
<receiver
    android:name="com.google.android.gcm.GCMBroadcastReceiver"
    android:permission="com.google.android.c2dm.permission.SEND">
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        <action android:name="com.google.android.c2dm.intent.REGISTRATION" />
        <category android:name="my_app_package" />
    </intent-filter>
</receiver>
```
GCMIntentService

// called after GCM library finishes registration
// you need to send regId to your server
onRegistered(Context context, String regId);

onUnregistered(Context context, String regId);

// called after your server sends a msg to GCM, and
// GCM sends to this device
onMessage(Context context, Intent intent);

.onError(Context context, String errorId);
onRecoverableError(Context context, String errorId)
If you use GCM server library

```java
import com.google.android.gcm.server.*;

Sender sender = new Sender(myApiKey);
Message message = new
    Message.Builder().build();
MulticastResult result =
    sender.send(message, devices, 5);
```
Summary: GCM Flow

- Enabling cloud to device messaging
  - App (on device) registers with Google to get registration ID
  - App sends registration ID to its App Server

- Per message
  - App Server sends (authenticated) message to Google
  - Google sends message to device, which sends to app

- Disabling cloud to device messaging
  - App can unregister ID, e.g., when user no longer wants push
Additional Details

Discussion: Mobile Cloud Services

- We have discussed push notification service. What other services can you think of?
Example Mobile Cloud Services

- Push notification service
- Location based service
  - Track service (supporting location based services)
- Storage and sync
  - Syncing and storage service (iCloud)
- Proxy service (Kindle Split Browser)
- Recognition services
  - Speech to text/text to speech service
  - Natural language processing service (open Siri API for 3rd party applications in the future)
Outline

- Admin
- Mobile cloud service
  - Push notification service
  - Track service
    - StarTrack Next Generation: A Scalable Infrastructure for Track-Based Applications, by Maya Haridasan, Iqbal Mohomed, Doug Terry, Chandramohan A. Thekkath, and Li Zhang, in OSDI 2010.
Many phones already have the ability to determine their own location:
- GPS, cell tower triangulation, or proximity to WiFi hotspots

Many mobile applications use location information.

Courtesy: Maya et al.
A Common Abstraction: Track

Time-ordered sequence of location readings

Latitude: 37.4013
Longitude: -122.0730
Time: 07/08/10 08:46:45.125
Application: Personalized Driving Directions

Goal: Find directions to new gym

≈ Take US-101

1. Depart Kirk Ave toward Balsam Ave < 0.1 mi
2. Turn right onto E Arbor Ave 0.1 mi
3. Turn left onto Borregas Ave, and then immediately turn right onto W Maude Ave 0.3 mi
4. Turn right onto N Mathilda Ave VALERO on the corner 0.7 mi
5. Take ramp right for US-101 North / Bayshore Fwy toward San Francisco 2.6 mi
6. At exit 399A, take ramp right and follow signs for Shoreline Blvd 0.6 mi
7. Turn left onto N Shoreline Blvd 1.5 mi
8. Turn right onto California St 0.8 mi
9. Arrive at near 2012 California St, Mountain View, CA 94040 on the right
   The last intersection is Escuela Ave
   If you reach S Rengstorff Ave, you’ve gone too far

Courtesy: Maya et al.
## A Taxonomy of Applications

<table>
<thead>
<tr>
<th></th>
<th>Personal</th>
<th>Social</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Current location</strong></td>
<td>Driving directions, Nearby restaurants</td>
<td>Friend finder, Crowd scenes</td>
</tr>
<tr>
<td><strong>Past locations</strong></td>
<td>Personal travel journal, Geocoded photos</td>
<td>Post-it notes, Recommendations</td>
</tr>
<tr>
<td><strong>Tracks</strong></td>
<td>Personalized Driving Directions, Track-Based Search</td>
<td>Ride sharing, Discovery, Urban sensing</td>
</tr>
</tbody>
</table>
StarTrack System

Insertion Application
Location Manager  ST Client

Insertion

ST Client
Application

ST Server

• Retrieval
• Manipulation
• Comparison
...
System Challenges

1. Handling error-prone tracks

2. Flexible programming interface

3. Efficient implementation of operations on tracks

4. Scalability and fault tolerance
Challenges of Using Raw Tracks

Advantages of Canonicalization:

- More efficient retrieval and comparison operations
- Enables StarTrack to maintain a list of non-duplicate tracks
StarTrack API: Track Collections

Creation

- TC MakeCollection(GroupCriteria criteria, bool removeDuplicates)

Manipulation

- TC JoinTrackCollections (TC tCs[], bool removeDuplicates)
- TC SortTracks (TC tC, SortAttribute attr)
- TC TakeTracks(TC tC, int count)
- TC GetSimilarTracks (TC tC, Track refTrack, float simThreshold)
- TC GetPassByTracks (TC tC, Area[] areas)
- TC GetCommonSegments(TC tC, float freqThreshold)
API Usage: Ride-Sharing Application

// get user’s most popular track in the morning
TC myTC = MakeCollection("name = Maya", [0800 1000], true);
TC myPopTC = SortTracks(myTC, FREQ);
Track track = GetTracks(myPopTC, 0, 1);

// find tracks of all fellow employees
TC msTC = MakeCollection("name.Employer = MS", [0800 1000], true);

// pick tracks from the community most similar to user’s popular track
TC similarTC = GetSimilarTracks(msTC, track, 0.8);
Track[] similarTracks = GetTracks(similarTC, 0, 20);

// Verify if each track is frequently traveled by its respective owner
User[] result = FindOwnersOfFrequentTracks(similarTracks);
Track Similarity

SIM(A,B) = |S1–5| / |S1–5| / |S6–7|

SIM(A,C) = |S1–4| / |S1–4|+ |S5|+ |S6–7|
Summary

- The Track abstraction is simple but quite interesting

- Think about abstractions in your project