High-performance Android apps

Tim Bray

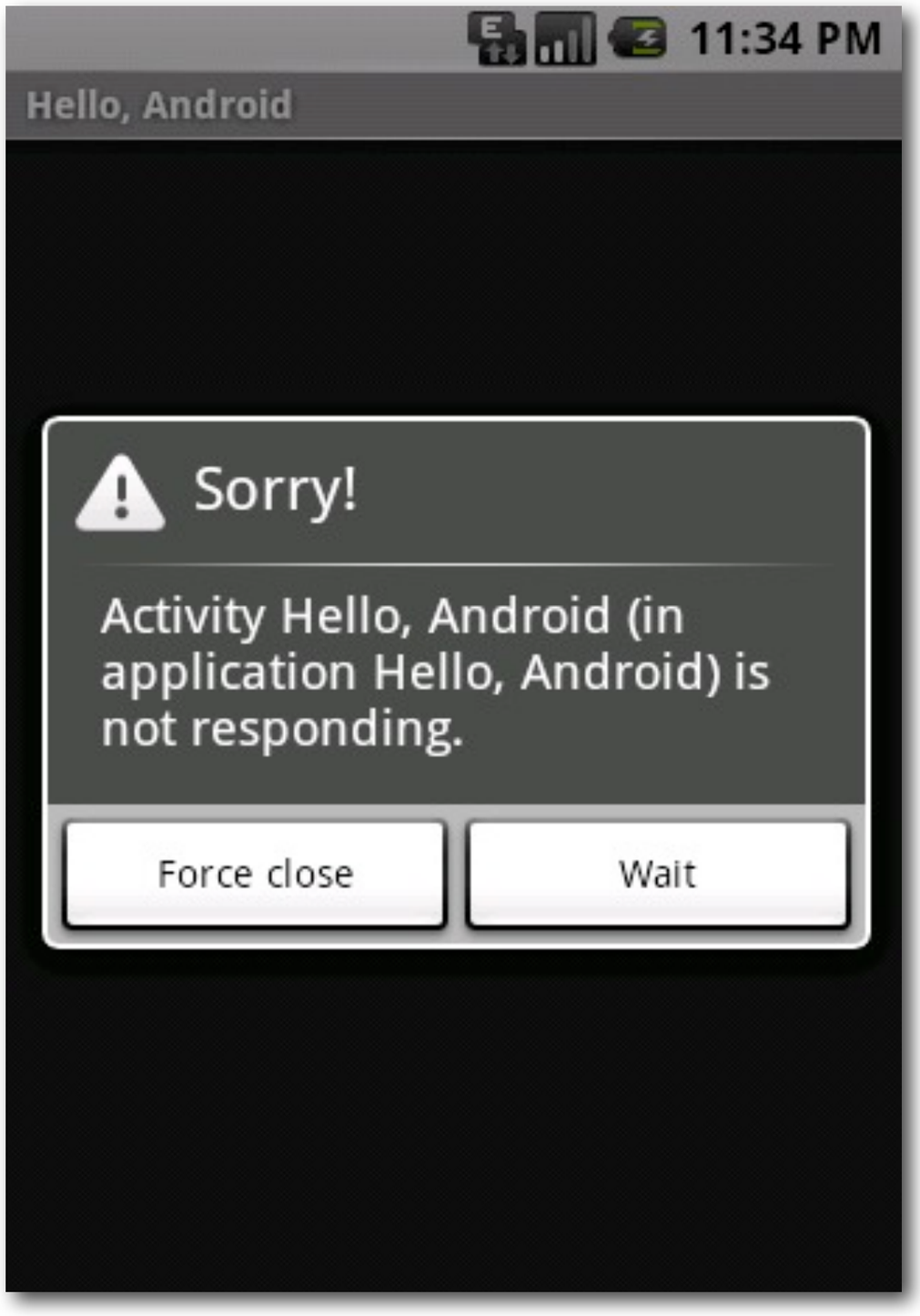
Two Questions

How do you make your app run fast?

When your app has to do some real work, how do keep the user experience pleasant?

“Jank”

- Chrome team's term for stalling the event loop, i.e. not instantly responsive to input; now in Android we say “A janky app”
- Eliminate by:
  - Reacting to events quickly
  - Don't hog the event loop (“main” / UI) thread!
  - Getting back into the select() / epoll_wait() call ASAP, so you can react to future events quickly (touches, drags)
- Else...


11:34 PM
Hello, Android
Sorry!
Activity Hello, Android (in application Hello, Android) is not responding.
Force close
Wait

ANR: “App Not Responding”

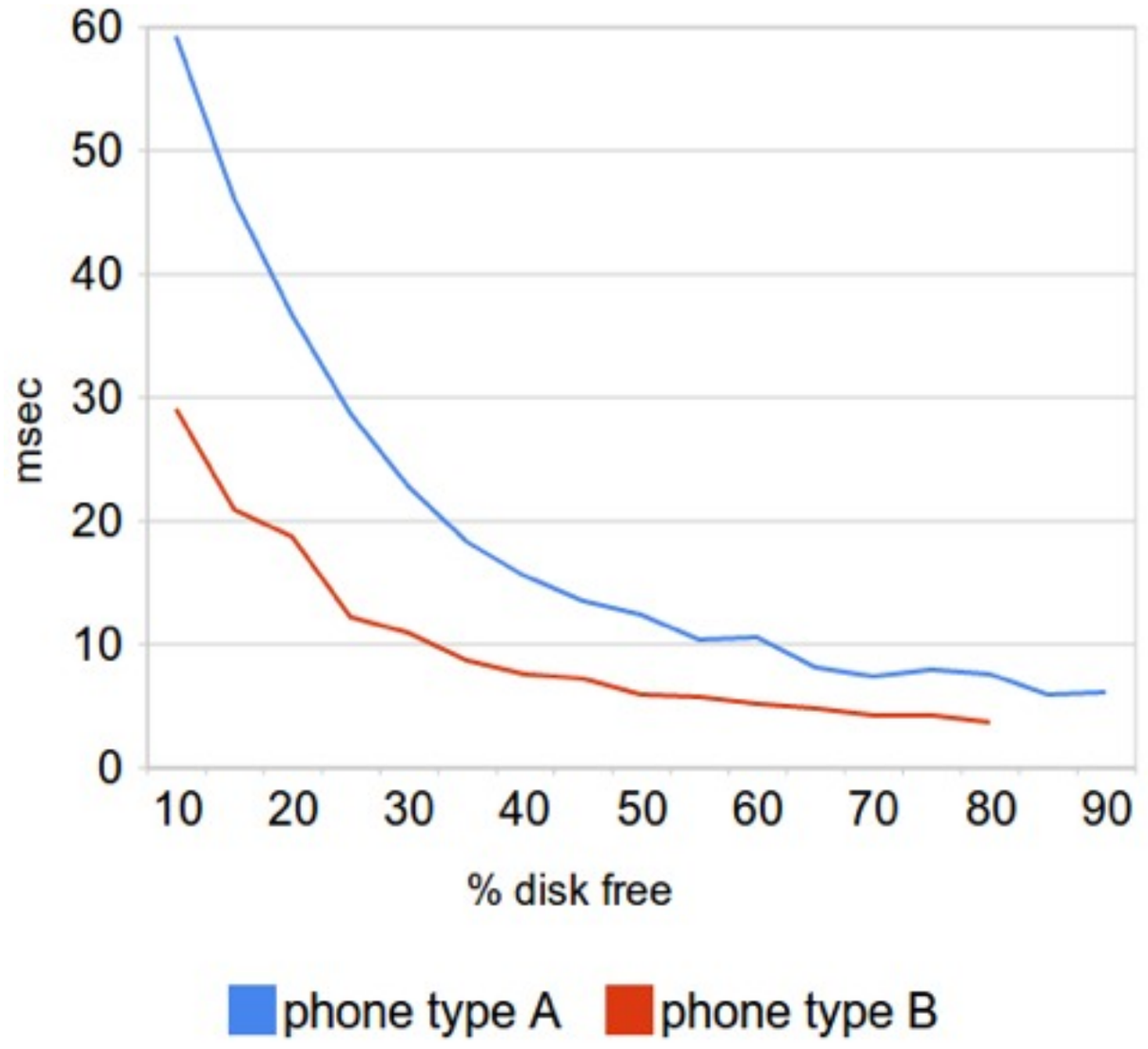
- Happens when:
  - UI thread doesn't respond to input event in 5 seconds, or
  - a BroadcastReceiver doesn't finish in 10 seconds
    - Typically due to network or storage operations on main thread
- But users complain about delays *much* less than 5 seconds!

Some Nexus One Numbers

One frame of 60 fps video: 16 ms. Human perception of slow action: 100-200 ms.

- ~0.04 ms: writing a byte on pipe process A->B, B->A or reading simple /proc files from Dalvik
- ~0.12 ms: void/void Binder RPC call A->B, B->A
- ~5-25 ms: uncached flash reading a byte
- ~5-200+**(!)** ms: uncached flash writing tiny amount
- 108/350/500/800 ms: ping over 3G. Variable!
- ~1-6+ seconds: TCP setup + HTTP fetch of 6k via 3G

Writing to flash (yaffs2)



Source: empirical samples over Google employee phones (Mar 2010)

- Create file, 512 byte write, delete (ala sqlite .journal in transaction)
- Flash is different than disks you're likely used to: read, write, erase, wear-leveling, GC
- Write performance is *highly variable!*

Sqlite Performance

- There’s no such thing as a cheap write
- Use indexes (see EXPLAIN & EXPLAIN QUERY PLAN)
- For logging, consider file-append rather than database-write

Lessons

- Writing to storage is slow
- Using the network is slow
- Always assume the worst; performance is guaranteed to produce bad reviews and Market ratings

Lessons

- Writing to storage is slow
- Using the network is slow
- Always assume the worst; performance is guaranteed to produce bad reviews and Market ratings
- So do your I/O and heavy computation on another thread, not the UI thread!

Tools: asyncTask

"AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers."

Tool: asyncTask

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {  // on some background thread
        int count = urls.length; long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
        }
        return totalSize;
    }
    protected void onProgressUpdate(Integer... progress) {  // on UI thread!
        setProgressPercent(progress[0]);
    }
    protected void onPostExecute(Long result) {  // on UI thread!
        showDialog("Downloaded " + result + " bytes");
    }
}
new DownloadFilesTask().execute(url1, url2, url3);  // call from UI thread!
```

Tool: asyncTask

“Fire and forget” mode

```
private boolean handleWebSearchRequest(final ContentResolver cr) {
      ...
      new AsyncTask<Void, Void, Void>() {
        protected Void doInBackground(Void... unused) {
          Browser.updateVisitedHistory(cr, newUrl, false);
          Browser.addSearchUrl(cr, newUrl);
          return null;
        }
      }.execute()
      ...
      return true;
  }
```

asyncTask Details

- Must be called from a main thread
  - rather, a thread with a Handler/Looper
  - No nested calls!
- An activity process may exit before its AsyncTask completes (user goes elsewhere, low RAM, etc).
- If this is a problem, use IntentService

Tool: android.app.IntentService

- “IntentService is a base class for Services that handle asynchronous requests (expressed as Intents) on demand. Clients send requests through startService(Intent) calls; the service is started as needed, handles each Intent in turn using a worker thread, and stops itself when it runs out of work.”
- Intent happens in a Service, so Android tries hard not to kill it
- Easy way to do use a Service

Calendar's use of IntentService

```
public class DismissAllAlarmsService extends IntentService {
  @Override public void onHandleIntent(Intent unusedIntent) {
    ContentResolver resolver = getContentResolver();
    ...
    resolver.update(uri, values, selection, null);
  }
}
```

In **`AlertReceiver extends BroadcastReceiver, onReceive()`** (main thread)

```
Intent intent = new Intent(context, DismissAllAlarmsService.class);
context.startService(intent);
```

UI Tips

- Disable UI elements immediately, before kicking off your AsyncTask to finish the task
- Use an animation or ProgressDialog to show you're working
- One example strategy:

1. Immediately, disable UI elements
2. Briefly, a spinner in the title bar
3. If more than 200msec, show a ProgressDialog
4. in AsyncTask onPostExecute, cancel alarm timer

App performance lessons

- Network and storage UI hurt (but you already knew that)
- Excessive memory allocation hurts
- Reflection hurts
- See “Design for Performance” at developer.android.com: Static is good, internal getters/setters hurt, floats and enums hurt, make constants final,

Demo: LifeSaver

- Suppose you're getting a new Android phone. Your contacts and calendar and so on will move over automatically.
- But your history of phone calls and text messages won't. This is your life!
- So, run LifeSaver on your old phone and it copies that history to the SD card.
- Move the SD card to the new phone and run it again and it copies the history back.
- This takes a while!

Performance Advice

"Premature optimization is the root of all evil."
-Donald Knuth

Performance Advice

"Premature optimization is the root of all evil."
-Donald Knuth

1. Design the simplest thing that could possible work, where "could possibly work" excludes doing network transactions on the UI thread.
2. If it's fast enough, you're done!
3. If it's not fast enough, **don't guess**. Measure and find out why.
4. Fix the biggest performance problems.
5. goto 2

Profiling Tools

- Traceview
- Log.d() calls with a timestamp aren't terrible
- Extreme profiling: Aggregate user profile data

Traceview

1. android.os.Debug.startMethodTracing(String fname)
2. android.os.Debug.stopMethodTracing()
3. adb pull /sdcard/<fname>.trace <fname>.trace
4. traceview <fname>

Demo

Profiling Tim's "LifeSaver 2" application with traceview.

Traceview: Cautions

1. Overestimates the penalty for method dispatch
2. JIT is disabled
3. Numbers are not quantitatively reliable
4. Really only gives you a general feeling for what's happening

Future directions

1. Optimized screen manager
2. “Strict” mode, to detect I/O on the UI thread
3. Better profiling tools!

Thank you!

feedback: bit.ly/mgddbr
Tim Bray, Developer Advocate
twbray@google.com android-developers.blogspot.com @AndroidDev