Performance Diagnosis of Cloud-Based Mobile Applications

Yu Kang
Supervised by Prof. Michael Lyu
05/07/2016
Outline

• Introduction
• Client Side Performance Diagnosis
• Cloud Side Performance Diagnosis
• Conclusion
Complaints on Performance

• User complaints on the performance of mobile apps

Latest update seems to be taking longer to load. The old version worked better. I expected it to be faster and smoother. It's not. Touch screen is laggy. I don't like this.

I haven't had any issues with the last update. It seems more glitchy now. Super laggy all of a sudden. And now, on top of being laggy, all of my rubies are gone. I'm not impressed.

Not getting any better. The scrolling is awkward and clunky. It takes a long time too. Sometimes it just stop. Sometimes images load really slowly.

Buggy. Version 2.0 brings a well deserved change to the game but man is it buggy and laggy as hell. The animations are so choppy! Fix them soon they make the game less enjoyable. Trials and crusade are a lot harder by the way! I'm unable to clear difficulty 6 which I was able to do easily before the update.
Cloud-Based Mobile App

• Cloud service introduced
  - Mobile cloud market: $46.90 billion by 2019
• Typical framework (e.g., Google cloud endpoints)
Performance Still Unsatisfying

• Example: 12306
  – Official Chinese railway ticket booking app
  – Hundreds of millions users

Server side performance

Client side performance

Performance Diagnosis of Cloud-Based Mobile Applications
Performance Diagnosis for Cloud-Based Mobile App

• Diagnose performance on both client side and cloud side
Performance Diagnosis for Cloud-Based Mobile App

• On client side
  – Detect and diagnose performance issues
  – Enhance user experience for long executing operations

• On cloud side
  – Reduce cloud-client communication delay
  – Reduce cross-(data) center communication delay

• All tools, source codes, data released
Outline

• Introduction
• Client Side Performance Diagnosis
  – Android Performance Diagnosis via Anatomizing Asynchronous Executions (DiagDroid)
  – Detecting Poor Responsiveness UI for Android Applications (Pretect)
• Cloud Side Performance Diagnosis
• Conclusion
Background

• C

• A

Performance Diagnosis of Cloud-Based Mobile Applications
Android Application Specifics

• User-interface (UI) oriented
  – UI thread = main thread
  – UI thread kept responsive (non-blocking)

• Asynchronous executions → User perceived latency
  – Time-consuming tasks
  – Worker threads
  – Update UI afterwards
public class MyActivity extends Activity {
    private class RetrieveDataTask extends AsyncTask<String, Void, String> {
        ... 
        protected String doInBackground(String... urls) {
            // Retrieve content from Internet
            return content;
        }
        protected void onPostExecute(String content) {
            this.textView.setText(content);
        }
    }
    private class MyOnClickListener implements OnClickListener {
        @Override
        protected void onClick(View v) {
            retrieveDataTask task1, task2, task3;
            task1 = new retrieveDataTask(textView1);
            task2 = new retrieveDataTask(textView2);
            task3 = new retrieveDataTask(textView3);
            task1.execute(url1);
            task2.execute(url2);
            task3.execute(url3);
        }
    }
}
Performance Issue - Sequential Running

• Goal: load contents in parallel
  – Straight-forward approach
    • Call default `execute` method 3 times
    • Wrong!
    • Reason: tasks in a global queue
  – Correct approach
    • Resort to a thread pool
    • Call `executeOnExecutor` method instead

– Existing tool (e.g., StrictMode, Asynchronizer) cannot detect such bugs

private class MyOnClickListener implements OnClickListener {
    @Override
    protected void onClick(View v) {
        retrieveDataTask task1, task2, task3;
        task1 = new retrieveDataTask(textView1);
        task2 = new retrieveDataTask(textView2);
        task3 = new retrieveDataTask(textView3);
        // the frist trial on executing tasks in parallel
        if (Build.VERSION.SDK_INT >=
            Build.VERSION_CODES.HONEYCOMB) {
            task1.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, url1);
        } else {
            task1.execute(url1);
        }
    }
}
A Motivating Example in Bug Detection

- Difference: queuing time, pool capacity
DiagDroid Framework

• Two key modules
  − Profiler
  − Log analyzer

• Two parts
  − PC & Mobile

• Two mechanisms
  − Static & Dynamic Analysis
Profiler

• Features to profile
  – Runtime info (e.g., Queuing time, Execution time)
  – Identification info (e.g., Execution context (call-stack), pool identifier)
Profiler

• Challenges
  1. Tremendous ways to run asynchronous executions
  2. Android fragmentation
  3. Low overhead

• Solutions
  1. Android asynchronous execution taxonomy
  2. Hook only general framework methods
    • E.g., For ThreadPoolExecutor execute, beforeExecute, and afterExecute methods
  3. Granularity
    • Task level vs. method/line level

Performance Diagnosis of Cloud-Based Mobile Applications
Log Analyzer

- Target
  - Parse logs for statistics
  - Find anomaly in statistics

- Challenge
  - Too many similar contexts (call stacks)

- Solution
  - Cluster similar contexts

Performance Diagnosis of Cloud-Based Mobile Applications
Experimental Study

• Test configuration
  – 4 devices
  – 4 types of pressures + 1 without pressure

• 30 minutes test per configuration

• Totally 19,800 minutes test for 33 apps
Bugs Found

• Found: 27 new bugs of 5 types in 33 open source apps
  1. Sequential execution
  2. Forgetting cancelling execution
  3. Improper execution pool
  4. Message queue overloading
  5. Misusing third-party library

• Note:
  – No a priori knowledge on the app
A Diagnosis Example

• App: Transportr
• Report: **long queuing time** (>= 500ms)
• Details:
  − Short queuing tasks - avg. queue size: 0.04, avg. exec time in queue: 63.83ms
  − Long queuing tasks - avg. queue size: 1.38, avg. exec time in queue: 817.44ms
  − Pool capacity: 1
  − Context: performFiltering method of class LocationAdapter
• Root cause:
  − Overload message queue: **blocking** asynctask in handler
  − Developers confirmed and fixed in new versions
Summary of DiagDroid

• New type of performance issues
• Novel diagnosing framework: DiagDroid
• Categorize asynchronous executions
• New bugs found
• http://www.cudroid.com/DiagDroid

Performance Diagnosis of Cloud-Based Mobile Applications
Outline

• Introduction

• Client Side Performance Diagnosis
  – Android Performance Diagnosis via Anatomizing Asynchronous Executions (DiagDroid)
  – Detecting Poor Responsiveness UI for Android Applications (Pretect)

• Cloud Side Performance Diagnosis

• Conclusion
Introduction

• User expected waiting time
• Previous work 27 performance issues out of 48 reported
  – Resource limitation
• UI feedback required to enhance user experience

Latest update is okay, but still horrible. It seems that you can't revert it back to last version, but that okay. One problem is loading... I hate longer waiting than expected waiting time. Even, I keep touch somewhere and no response for 5 or 10 seconds. That's LAGGY. Period. Try again... I love this game so much.
Motivation

- Android unique UI mechanism
  - Activity not responding (ANR)
  - Asynchronous tasks & UI update

- UI responsiveness
  - Simple code may contain non-responsive UI design

```java
private class ImageDownloader extends AsyncTask<String, Void, Bitmap> {
    protected Bitmap doInBackground(String... urls) {
        return downloadBitmap(urls[0]);
    }

    protected void onPostExecute(Bitmap result) {
        imageView.setImageBitmap(result);
    }
}
```
Introduction

• Poor-responsive UI
  – long executing without feedback

• Challenge
  1. Hard to obtain feedback delay
  2. Threshold not clear
  3. Impossible to design feedback for all operations

• Our contribution
  – Real world user study (For Challenge 2)
  – A tool (Pretect) that assists delay tolerant UI design (For Challenge 1 & 3)
User Study

• Impatient mobile users
• Study relationship between users experience & operation delay
• Test settings
  – Three delay levels (200ms, 500ms, 2000ms)
  – Between-subject test (i.e., fixed delay level per user)
  – Compare overall performance rating
• Results
  – (Relationship) User experience & UI responsiveness
  – (Threshold) 500ms no response is lag enough
Problem Specification

• Operation feedback
  – First screen update after a user operation

• Feedback delay
  – Latency between the input event and the feedback

• Poor-responsive operations
  – Operations with feedback delay $\geq T$ ($T=500\text{ms}$)
Challenging Task

• Task: detecting poor-responsive operations

• Current tool
  − Detecting abnormal tasks despite feedback delay (e.g., DiagDroid)

• First Trial
  − Monitoring all display updates

• Solution
  − Monitor UI update procedure
  − Separate system display update (e.g., notification bar) with app UI updates

Performance Diagnosis of Cloud-Based Mobile Applications
Framework

• Execution flow
  1. Record user inputs
  2. Capture display updates
  3. Analyze the feedback delay

• Corresponding modules
  1. Event monitor
  2. Display monitor
  3. Log analyzer
Implementation

• Highlights:
  − Compatibility: dynamic instrumentation mechanism
  − Usability: no recompiling of OS/framework/app & easy to install
  − Android specifics: JAVA hook – Zygote, C hook – ptrace

• Event monitor
  − Instrument framework Java methods
  − Sample log:

    ... 2365 ...: com.cyberlink.youperfect[Event]com.cyberlink.youperfect.widgetpool.common.ChildProportionLayout(425193c8V.E...C....P....270,0-540,67#7f0a051dapp-id/cutout_tab_artistic).null-Motion-UP : 125696

Performance Diagnosis of Cloud-Based Mobile Applications
Implementation

• Display monitor
  – Hook C inter-process communication (IPC) to surfaceflinger
  – Sample log:
    ```
    ... 138 ...: BIPC:****android.gui.SurfaceTexture****, sender_pid:2365, UptimeMilli: 127932
    ```

• Log analyzer
  – Correlate input events and UI updates via checking the source *pid* of both
  – Compute feedback delay and report poor-responsive operations
Case Study

• YouCam
  – Popular selfie app over 60 million downloads
  – Confirmed & fixed by the developer
  – “With your hint, we find that we have used a widget which tends to be slower. We will fix as soon as possible.”

YouCam - steps to reproduce

1. Open YouCam
2. Open “Cutout”
3. Click “Artistic”
4. Click “Fun” tab
5. Select the first cover
6. Select the first cover
7. Click “tick” image
8. Randomly draw a line
9. Click “tick” image
10. Switch to “Artistic”
11. Switch to “Fun”
12. Select the first cover
13. Click the “save” text

YouCam – sample reports

Performance Diagnosis of Cloud-Based Mobile Applications 32
Summary of Pretect

- Real-world user study
- A tool (Pretect)
- Cases detected
- http://www.cudroid.com/pretect

Performance Diagnosis of Cloud-Based Mobile Applications
Outline

• Introduction
• Client Side Performance Diagnosis
  – Deployment of Single Cloud Service
  – Deployment of Multiple Cloud Services
• Conclusion
Cloud-Service Involved

- Cloud-based mobile app
- Modeling User Experience
  - User-data center delay (UC delay)
- Importance of service deployment
Optimizing UC Delay

• Challenges
  • Not every data center visited
  • Difficult to foresee user experience

• Solution
  – Measure UC delay: recorded round-trip time (RTT)
  – Predict UC delay: delay prediction according to similar users
Framework of Cloud-Based Services

• Measure UC delay
• Predict UC delay
Minimize Average Cost

Given:
Z = set of data centers
C = set of users
d_{ij} = pairwise distance (i,j) ∈ C × Z

Minimize:
\[ \sum_{i=1}^{N} \min_{T \in \mathcal{E}} \{ d_{ij} \} \]

Subject to:
\[ Z' \subseteq Z, |Z'| = k \]
Minimize Average Cost

k-median problem
Problems of the Model

• Unnecessary minimum
• Outlier users
• Tradeoff
  – (Response time) ≤ threshold $T$
  – (User number) 99% good, 1% poor
Maximize Close User Amount

Given Bipartite graph $B(V_{1,2}, E)$ where

$\mid V_1 \mid = M, \mid V_2 \mid = N$

$i \in V_1, j \in V_2$

$\{i,j\} \in E, d_{ij} \leq T \ (i,j) \notin E, \text{ otherwise.}$

Maximize:

$\mid N_B(V') \mid$

Subject to:

$V' \subset V_1, \mid V' \mid = k$
Maximize Close User Amount

\{v_1, v_2, v_3, v_5\} \cap \{v_1, v_2, v_4\} \cap \{v_1, v_3, v_4\} \cap \{v_4, v_5\}

Max k-cover problem

Data center
User

Performance Diagnosis of Cloud-Based Mobile Applications
Real-World Dataset

- 303 PlanetLab computers
- 4,302 the Internet services
- $\approx 130,000$ response-time values matrix
Outline

• Introduction
• Client Side Performance Diagnosis
  – Deployment of Single Cloud Service
  – Deployment of Multiple Cloud Services
• Cloud Side Performance Diagnosis
• Conclusion
Background

• Extending previous model
• Different cloud services may cooperate
  – YouTube & Facebook
  – Google Doc & Gmail
  – Taobao & Alipay
• Necessary to deploy together
  – Independent deployment not enough
  – Global decision required
Motivation Example
Multi-Service Co-deployment Problem

- Same company to host
- **Multiple** services for different users (may overlap)
- Interaction between services

Another model for deploying simultaneously!
Modeling Performance Diagnosis of Cloud-Based Mobile Applications

\[
\begin{align*}
\text{minimize} & & \sum_{\substack{h \leq s \leq m, j \in C_h}} r_{hs} d_{hs} x_{hsj} + \sum_{\substack{i \in U, q \neq s, r \in C_s}} r_{iqr} d_{iqr} y_{ipqrs} \\
\text{subject to:} & & \\
& & \sum_{j \in C_h} x_{hsj} = \text{sign}(r_{hs}), \quad 1 \leq h \leq m, \forall i \in U, \\
& & x_{hsj} \leq z_{hsj}, \quad 1 \leq h \leq m, \forall j \in C_h, \\
& & \sum_{j \in C_h} z_{hsj} \leq k_h, \quad 1 \leq h \leq m, \\
& & \sum_{\substack{1 \leq s \leq m, q \neq s, \forall p \in C_q, \forall r \in C_s}} y_{ipqrs} \leq x_{hsj}, \quad 1 \leq h \leq m, \forall j \in C_h, \\
& & y_{ipqrs} \leq z_{rs}, \quad \forall i \in U, \quad 1 \leq q, s \leq m, q \neq s, \\
& & x_{hsj} \in \{0, 1\}, \quad 1 \leq h \leq m, j \leq C_h, \\
& & y_{ipqrs} \in \{0, 1\}, \quad 1 \leq q, s \leq m, q \neq s, \\
& & z_{hsj} \in \{0, 1\}, \quad \forall i \in U, \quad 1 \leq h \leq m, \forall j \in C_h.
\end{align*}
\]
Real-World Dataset

1. 597 Planetlab instances
2. Ping 2,213 web services
3. Ping all other Planetlab peers (random order)
4. Obtain ≈577,000 Internet-service access values matrix & ≈94,000 peer-wise communication delay values matrix
Summary of Cloud Service Deployment

• Model user experience
• Formulate deployment problems
• Real-world dataset
• http://appsrv.cse.cuhk.edu.hk/~ykang/cloud
Outline

• Introduction
• Client Side Performance Diagnosis
• Cloud Side Performance Diagnosis
• Conclusion
Conclusion

• Cloud-based mobile app performance enhancing
• On client side
  – Detect and diagnose performance issues
  – Enhancing user experience for long executing operations
• On cloud side
  – Reduce cloud-client communication delay
  – Reduce cross-(data) center communication delay
• All tools, source codes, data released
Thank you!

Q & A
Backup Slides
DiagDroid
Performance issue 2 – forget cancelling

• Goal: deal with dead tasks
  – Straight-forward approach
    • Don’t do anything
    • Wrong!
    • Reason: tasks do not cancel automatically
  – Alternative approach
    • Cancel the task (e.g., downloading) via overriding onCancel method
    • Wrong!
    • Reason: onCancel is called after doInBackground, cannot cancel tasks
  – Correct approach
    • Check isCancelled() periodically
    • Cancel whenever the function returns true

– Existing tool (e.g., StrictMode, Asynchronizer) cannot detect such bugs
public class MyActivity extends Activity {
    private class RetrieveInfoTask extends AsyncTask<String, Void, String> {
        ...
        protected String doInBackground(String... urls){
            ...
            while (!isCancelled() && (length = is.read(buf)) != -1) {
                ...
            }
        }
    }
    private RetrieveInfoTask task1, task2, task3;
    ...
    @Override
    public void onStop() {
        if (task1 != null) task1.cancel(true);
        if (task2 != null) task2.cancel(true);
        if (task3 != null) task3.cancel(true);
        super.onStop();
    }
}
Detect bug in Motivating Example

• Profiler:
  – Profile tasks
  – Queuing time, executing time, task context (call-stack), pool info, and etc

• Log analyzer:
  – Find the problematic task
  – Queuing time, executing time, pool conflicts
Profiling mechanisms (example)

• Profiling ThreadPool:
  – Execution context – call stack when invoking execute method of the ThreadPoolExecutor class
  – Pool id – hash code of the thread pool
  – Request time – same time when collecting context
  – Start time & End time – time invoking beforeExecute & afterExecute method

• Profiling AsyncTask:
  – Execution context – call stack when invoking execute or executeOnExecutor method of the AsyncTask class
  – Pool id – hash code of the relevant pool(s)
  – Request time – the time when invoking execute or executeOnExecutor method of the AsyncTask class
  – Start time & End time – reuse ThreadPool mechanism
Log Analyzer

• Target
  – Parse logs for statistics
  – Find anomaly in statistics

• Challenge
  – Too many similar contexts (call-stacks)

• Solution
  – Cluster similar contexts
  – Detect anomalous by maximum (> threshold T)

Performance Diagnosis of Cloud-Based Mobile Applications
Other modules

• Static analysis
  – Decompile the app and gather information from bytecode
  – Do not modify the original app

• Test executor
  – A guard of Monkey Exerciser (a random testing tool)
  – Support plugin of any kind of test scripts
## DiagDroid - Bugs found

- [www.cudroid.com/DiagDroid](http://www.cudroid.com/DiagDroid)

<table>
<thead>
<tr>
<th>Category</th>
<th>Issue description</th>
<th>Class@App</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential execution</td>
<td>Not waiting AsyncTask.execute() method results in undesired sequential execution</td>
<td><code>LawListFragment@OpenLaw</code></td>
<td>1/4</td>
</tr>
<tr>
<td></td>
<td>Loading tens of icons in sequence</td>
<td><code>AppListArrayAdapter@AFWall+</code></td>
<td>1/3</td>
</tr>
<tr>
<td>Forgetting canceling execution</td>
<td>Improper cancelation of asynchronous tasks</td>
<td><code>GetRouteFaceTask@BartRunnerAndroid</code></td>
<td>1/4</td>
</tr>
<tr>
<td></td>
<td>Not canceling obsolete queries when new query arrives</td>
<td><code>AsyncQueryTripsTask@Liberario</code></td>
<td>2/2</td>
</tr>
<tr>
<td>Improper thread pool</td>
<td>Failed to set optimal size of the thread pool</td>
<td><code>ZLAndroidImageLoader@FBReader</code></td>
<td>1/2</td>
</tr>
<tr>
<td></td>
<td>Use the same pool for loading app list and app icons</td>
<td><code>MainActivity@AFWall+</code></td>
<td>2/3</td>
</tr>
<tr>
<td>Overloading message queue</td>
<td>Posting various types of tasks (e.g., update progress, store back) to the same</td>
<td><code>ReadingFragment@PageTurner</code></td>
<td>3/9</td>
</tr>
<tr>
<td></td>
<td>backgroundHandler</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Executing Filter method of AutoComplete-TextView occupies the Handler of a public</td>
<td><code>LocationAdapter@Liberario</code></td>
<td>1/2</td>
</tr>
<tr>
<td></td>
<td>message queue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Missing third-party library</td>
<td>Not canceling the tasks implemented by third-party library, Android asynchronous</td>
<td><code>HeadlineComposerAdapter@OpenLaw</code></td>
<td>4/4</td>
</tr>
<tr>
<td></td>
<td>http client - loopj</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use the deprecated findall method of WebView class which causes blocking</td>
<td><code>MainActivity@Lucid Browser</code></td>
<td>1/5</td>
</tr>
</tbody>
</table>
DiagDroid – Fix bugs

Fix Transportr

Fix AFWall+
DiagDroid – Low Overhead

• 10,000 Monkey operations with DiagDroid on and off
• 200 ms interval between two operations
• Time command for CPU time
• 0.8% overhead
DiagDroid - Development Tips

1. Use private pool instead of public one when necessary.
2. Set reasonable pool size.
3. Use third-party library carefully.
5. Cancel when no longer needed.
6. Use proper type of asynchronous execution.
Pretect
User Study

• Results:
  - The mean value shows clear descending trend with delay level

<table>
<thead>
<tr>
<th>Delay level</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 ms</td>
<td>5.59</td>
<td>2.14</td>
<td>38</td>
</tr>
<tr>
<td>500 ms</td>
<td>4.68</td>
<td>1.80</td>
<td>37</td>
</tr>
<tr>
<td>2000 ms</td>
<td>4.24</td>
<td>2.43</td>
<td>41</td>
</tr>
<tr>
<td>Total</td>
<td>4.82</td>
<td>2.21</td>
<td>116</td>
</tr>
</tbody>
</table>

  - The Pairwise comparisons show the significance
    • 200ms vs. 500ms: marginal difference
    • 200ms vs 2000ms: Significant difference
    • 500ms vs. 2000ms: no much difference
Experiment

• Tool validation
  – Synthetic apps
    • Five common asynchronous execution mechanisms
  – Open source projects code injection
    • Common operations of five open source projects including delays incurred by db, network, remote process, disk operations
  – Result suggests we could distinguish (poor)-responsive operations
    • feedback delay ≤ 500ms

Performance Diagnosis of Cloud-Based Mobile Applications
Experiment

• Overall
  – Apply to 115 popular Android apps covering 23 categories (including BooksReferences, Photography, Sports, etc)

<table>
<thead>
<tr>
<th>App Statistics</th>
<th>Issues Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td># apps contain bugs</td>
<td>Total</td>
</tr>
<tr>
<td>94</td>
<td>327</td>
</tr>
<tr>
<td>Max bugs an app</td>
<td>Max</td>
</tr>
<tr>
<td>23</td>
<td>29189.0 (ms)</td>
</tr>
<tr>
<td>Min bugs an app</td>
<td>Min</td>
</tr>
<tr>
<td>0</td>
<td>504.0 (ms)</td>
</tr>
<tr>
<td>Avg. bugs per app</td>
<td>Avg</td>
</tr>
<tr>
<td>2.8</td>
<td>1603.9 (ms)</td>
</tr>
<tr>
<td>Median bugs per app</td>
<td>Stdv</td>
</tr>
<tr>
<td>2</td>
<td>2635.5</td>
</tr>
</tbody>
</table>
Experiment Stats

• 94/115 apps contain potential UI design defects
• 327 independent components with feedback delays ≥ 500 ms
• Maximum delay ≥ 29 s
Experiment Stats

Distribution of number of issues per app

Distribution of feedback delay per app
Different threshold

Avg. number of cases per category by threshold
## Poor-responsive Components

<table>
<thead>
<tr>
<th>Component</th>
<th>Issues found</th>
</tr>
</thead>
<tbody>
<tr>
<td>android.widget.Button</td>
<td>78</td>
</tr>
<tr>
<td>android.widget.ListView</td>
<td>30</td>
</tr>
<tr>
<td>android.widget.ImageButton</td>
<td>22</td>
</tr>
<tr>
<td>android.widget.EditText</td>
<td>21</td>
</tr>
<tr>
<td>android.widget ScrollView</td>
<td>20</td>
</tr>
<tr>
<td>android.widget.RelativeLayout</td>
<td>16</td>
</tr>
<tr>
<td>android.widget.ImageView</td>
<td>13</td>
</tr>
<tr>
<td>android.widget.TextView</td>
<td>10</td>
</tr>
<tr>
<td>android.widget.LinearLayout</td>
<td>10</td>
</tr>
<tr>
<td>android.support.v7.widget.Toolbar</td>
<td>7</td>
</tr>
</tbody>
</table>
Single Service Deployment
Introduction

Cloud Computing Systems

- **Auto scaling**
  Dynamic allocation of computing resources

- **Elastic load balance**
  Distributes and balances the incoming traffic
Introduction

• Typical approach of auto scaling and load balance (Amazon EC2)
Introduction

Current approaches are not optimized for users

- Auto scaling
  Do not consider distributions of the end users

- Elastic load balance
  Do not take the user specifics (*e.g.*, *user location*) into considerations
Introduction

• Our contribution:
  – User experience model in cloud
  – A new service redeployment method

• Two advantages:
  1) Improve auto scaling techniques
     Launch best set of service instances
  2) Extend elastic load balance
     Directs user request to a nearby one.
Minimize Average Cost

- k-median problem
- NP-hard
- $W[2]$-hard with $k$ as parameter
- $W[1]$-hard with capacity $l$ as parameter
- In FPT with both as parameter
  
  algorithm: $O(f(k,l)n^{o(1)})$ time
Minimize Average Cost

• Approximate Algorithms:
  1. Exhaustive Search
  2. Greedy Algorithm
  3. Local Search Algorithm (3 + \( \varepsilon \) approximation)
  4. Random Algorithm
Maximize Close User Amount

• **Max k-cover problem**
• NP-hard
• W[2]-hard with k as parameter
• W[2]-hard (general) and FPT (tree-like) with maximum subset size as parameter
• FPT if both maximum subset size and capacity as parameter
Maximize Close User Amount

• Approximate Algorithms:
  1. *Greedy Algorithm (1-1/e approximation)*
  2. *Local Search Algorithm*
Single Service – Necessity of Redeployment

• Worst case without redeployment
Single Service – Weakness of Auto Scaling

Deploy in limited data centers

Auto scaling algorithms
Single Service - Comparing Algorithms for k-Median

Selecting 3 data centers by redeployment algorithm

Selecting 10-20 data centers for 4000 users
### Single Service - Comparing Algorithms for k-Median

<table>
<thead>
<tr>
<th></th>
<th>Brute Force</th>
<th>Greedy</th>
<th>Greedy + init single swap</th>
<th>Random + init single swap</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2,200)</td>
<td>203</td>
<td>&lt; 1</td>
<td>&lt; 1</td>
<td>16</td>
</tr>
<tr>
<td>(2,400)</td>
<td>375</td>
<td>&lt; 1</td>
<td>16</td>
<td>31</td>
</tr>
<tr>
<td>(2,600)</td>
<td>547</td>
<td>&lt; 1</td>
<td>15</td>
<td>47</td>
</tr>
<tr>
<td>(2,800)</td>
<td>735</td>
<td>&lt; 1</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>(3,600)</td>
<td>78969</td>
<td>&lt; 1</td>
<td>31</td>
<td>63</td>
</tr>
<tr>
<td>(10,4000)</td>
<td></td>
<td>94</td>
<td>328</td>
<td>2641</td>
</tr>
<tr>
<td>(15,4000)</td>
<td></td>
<td>172</td>
<td>500</td>
<td>13109</td>
</tr>
<tr>
<td>(20,4000)</td>
<td></td>
<td>203</td>
<td>1906</td>
<td>25469</td>
</tr>
</tbody>
</table>

- **Theoretical time complexity**
  - **Exhaustive search:** $O(M^k \cdot N)$
  - **Greedy:** $O(k \cdot M \cdot N)$
  - **Local Search:** $O(k^t \cdot M^t \cdot N)$
Single Service - Redeployment Algorithms for Max k-Cover

• 20 instances are selected to provide service for 4000 users.
• Expect 200 per server.
Single Service - Redeployment Algorithms for Max k-Cover

Max k-cover using greedy approach

- compare the average cost: max k-cover v.s. k-median

Average cost by max k-cover model
Multi-Service Deployment
Independent Deployment of Single Service
Single Service Deployment

Minimize total distance for all user requests

\[
\text{minimize } \sum_{i \in U, j \in C} r_i d_{ij} x_{ij}
\]

subject to:

\[
\sum_{j \in C} x_{ij} = 1, \quad \forall i \in U,
\]

\[
x_{ij} \leq y_j, \quad \forall i \in U, j \in C,
\]

\[
\sum_{j \in C} y_j \leq k, \quad \forall i \in U,
\]

\[
x_{ij} \in \{0, 1\}, \quad y_j \in \{0, 1\},
\]

Times of user i call service

Distance between user i and VM j

Can only connect to open VMs

Every user i can only connect to one VM

Open at most k VMs

Indicator whether user i is connected to VM j

Indicator whether VM j is used
Independent Deployment of Single Service

Every user $i$ can only connect to one VM

Can only connect to open VMs

$x_{ij} = 1$

$y_j = 1$

Open at most $k$ VMs

Can only connect to open VMs

User
Instance

Data Center
Cost is recorded
Cost is predicted
Multi Cloud Service Co-deployment

Performance Diagnosis of Cloud-Based Mobile Applications

Times of user i call service h

\[ \sum_{j \in C_h} x_{hij} = \text{sign}(r_{hi}), \]

\[ x_{hij} \leq d_{hij}, \]

\[ \sum_{j \in C_h} z_{hij} \leq k_{hij}, \]

\[ \sum_{j \in C_h} y_{hij} \leq x_{hij}, \]

\[ y_{hij} \leq \text{sign}(r_{hi}), \]

\[ y_{hij} \leq z_{hij}, \]

\[ y_{hij} \leq t_{hij}, \]

both centers should be opened

Times of interaction between service q service s for request of user i

connect at most one center

limitation # of instances every service

at most one connection

indicator whether center j for service h is used

indicator whether interaction between services q and s go through center p and r for requests of user i
Motivation Example

First VM is chosen by user $i$ for service $h$, next is open

Open at most $k_h$ VMs for service $h$

Can only connect to open VMs

Connect at most one center
Iterative Sequential Co-deployment Algorithm

1. $tempS \leftarrow \phi, S \leftarrow \phi$
2. $temp \leftarrow MAX, tempmin \leftarrow MAX, min \leftarrow MAX$
3. for all service $h$ do
4. Select a set $S_h$ of $k_h$ service VMs randomly as candidate set $C_h$
5. $tempS \leftarrow tempS + S_h$
6. end for
7. for $i = 1 \rightarrow n$ do
8. $tempmin \leftarrow$ Evaluate the solution $tempS$
9. repeat
10. $temp \leftarrow tempmin$
11. for all service $h$ do
12. Select a set $S'_h$ of $k_h$ service VMs according to Model 1 with decided $tempS$
13. $tempS \leftarrow tempS - S_h + S'_h$
14. end for
15. $tempmin \leftarrow$ Evaluate the solution $tempS$
16. if $tempmin < min$ then
17. $min \leftarrow tempmin$
18. $S \leftarrow tempS$
19. end if
20. until $|tempmin - temp| \leq \epsilon$
21. Disturb the solution set $tempS$
22. end for
23. return $S$

First Generate Random Deployment

Sequentially improve the deployment of each service

Treat requests from other services as those from users

Record the best till now

Disturb and do local search
## Dataset Statistics

<table>
<thead>
<tr>
<th>Statistics</th>
<th>P2W Matrix</th>
<th>P2P Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimum</td>
<td>0.01</td>
<td>0.08</td>
</tr>
<tr>
<td>25th percentile</td>
<td>53.69</td>
<td>58.58</td>
</tr>
<tr>
<td>median</td>
<td>118.14</td>
<td>133.76</td>
</tr>
<tr>
<td>75th percentile</td>
<td>176.00</td>
<td>188.53</td>
</tr>
<tr>
<td>maximum</td>
<td>1604.02</td>
<td>5704.04</td>
</tr>
<tr>
<td>average</td>
<td>129.41</td>
<td>138.38</td>
</tr>
</tbody>
</table>
Experiment Setting

- Above $10^6$ decision variables
- Use the tool Ilog CPLEX provided by our department to solve the MIP problems
- Randomly generate user log and calling sequences as:
  - User id $\rightarrow$ service $s_{i1}$ $\rightarrow$ service $s_{i2}$ $\rightarrow$ ... $\rightarrow$ service $s_{im}$

<table>
<thead>
<tr>
<th>Notation</th>
<th>Descriptions</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_h$</td>
<td>Number of log entries of service $h$</td>
<td>1880</td>
</tr>
<tr>
<td>$\rho_h$</td>
<td>Ratio of users use service $h$ to total users</td>
<td>0.2</td>
</tr>
<tr>
<td>$l$</td>
<td>Number of services involved in one service request</td>
<td>5</td>
</tr>
</tbody>
</table>
Default Experiment Setting

- 1881 users
- 10 services
- Deploy 10 service VMs among a candidate set in 100 data centers
- A user of service $s$ would have 5 request logs
- One request of a service would involve on average 5 requests of other services
Experiment (Algorithm Specifics)

• *Convergence of Iterative Sequential Procedure*

![Convergence Graph](image1)

• *Number of Disturbs*

![Number of Disturbs Graph](image2)

Performance Diagnosis of Cloud-Based Mobile Applications
Experiment (*Number of Services*)

![Latency vs Service Number Graph]

- **Latency (ms)**
- **Service Number**

- Red line: Independent deployment
- Blue line: Iterative sequential algorithm
Experiment (Number of Service VMs)

- Size of Candidate Set of Service VMs

- Number of Service VMs to Deploy
Experiment (Services Logs)

- **Number of Service Users**

- **Average Call Length of Service**
Experiment (Services Logs)

- Number of Logs
Future work

• Mobile Side User Experience Enhancement
  – Selective Loading, Computing in Advance
  – Delay Tolerant UI Design

• Cloud Side Processing Time Optimization
  – Moving computation to data

• Client-Cloud Communication Cost Reduction
  – Reduce # communications


