Performance Diagnosis of Cloud-Based Mobile Applications

Yu Kang

Supervised by Prof. Michael Lyu

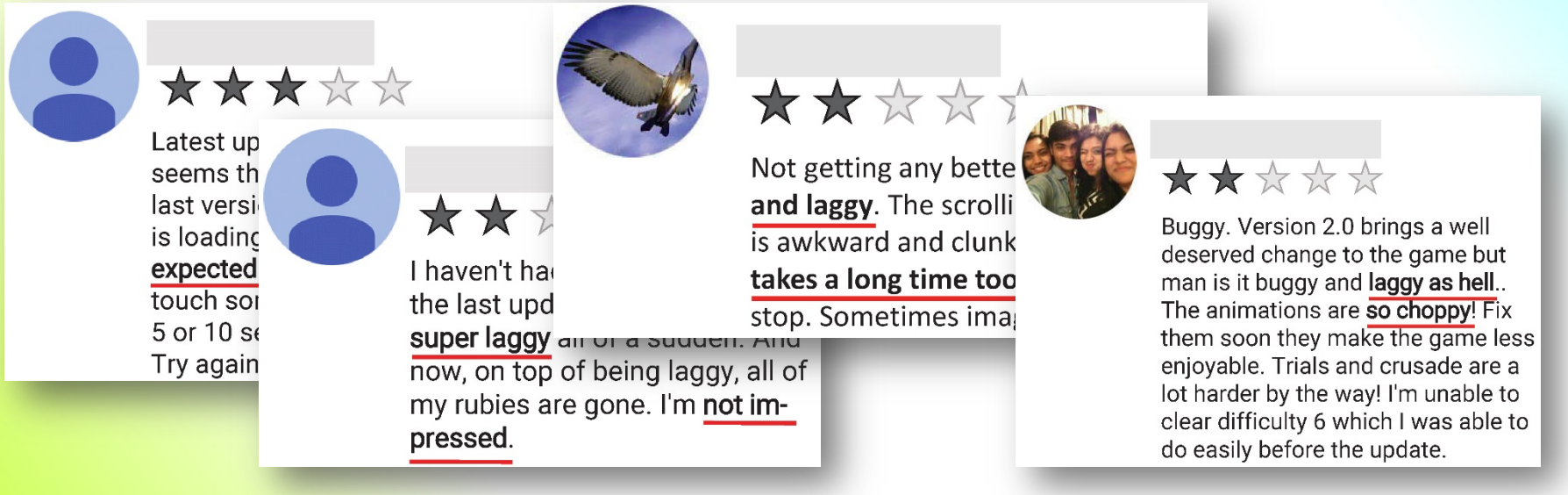
05/07/2016

Outline

- Introduction
- Client Side Performance Diagnosis
- Cloud Side Performance Diagnosis
- Conclusion

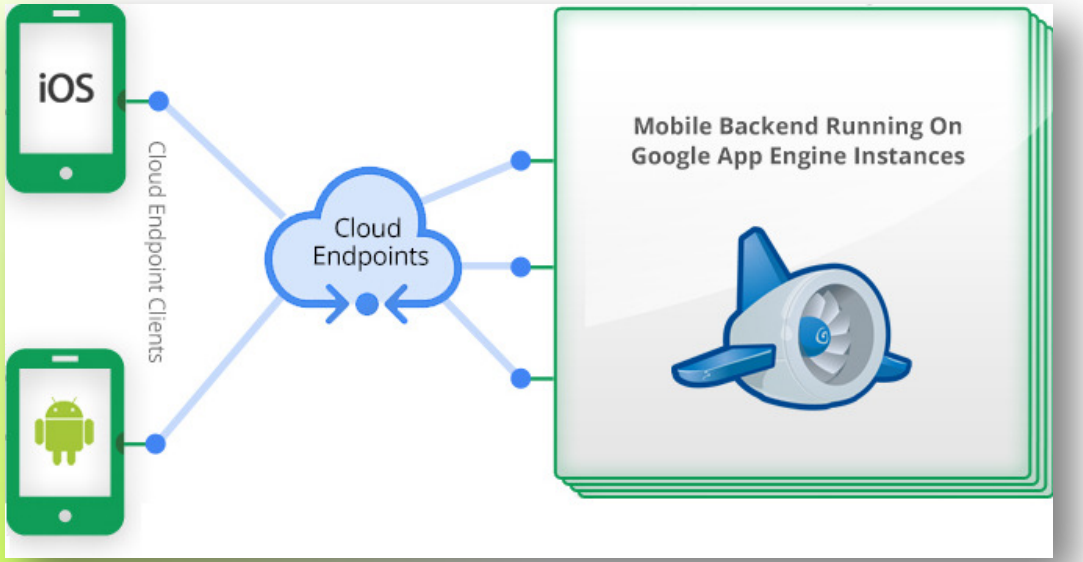
Complaints on Performance

- User complaints on the performance of mobile apps



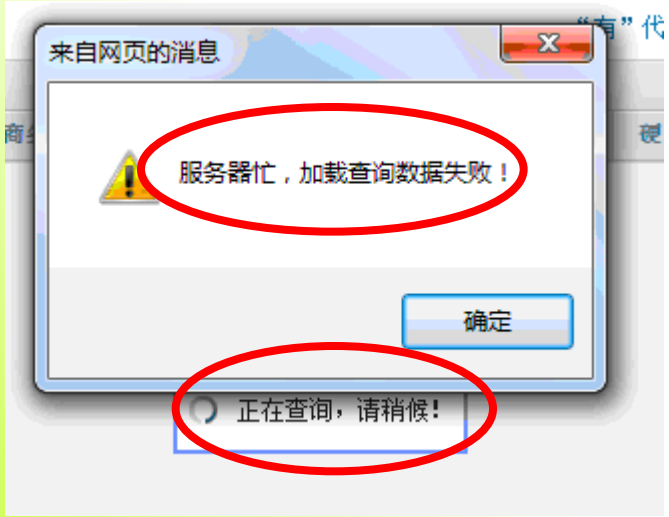
Cloud-Based Mobile App

- 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

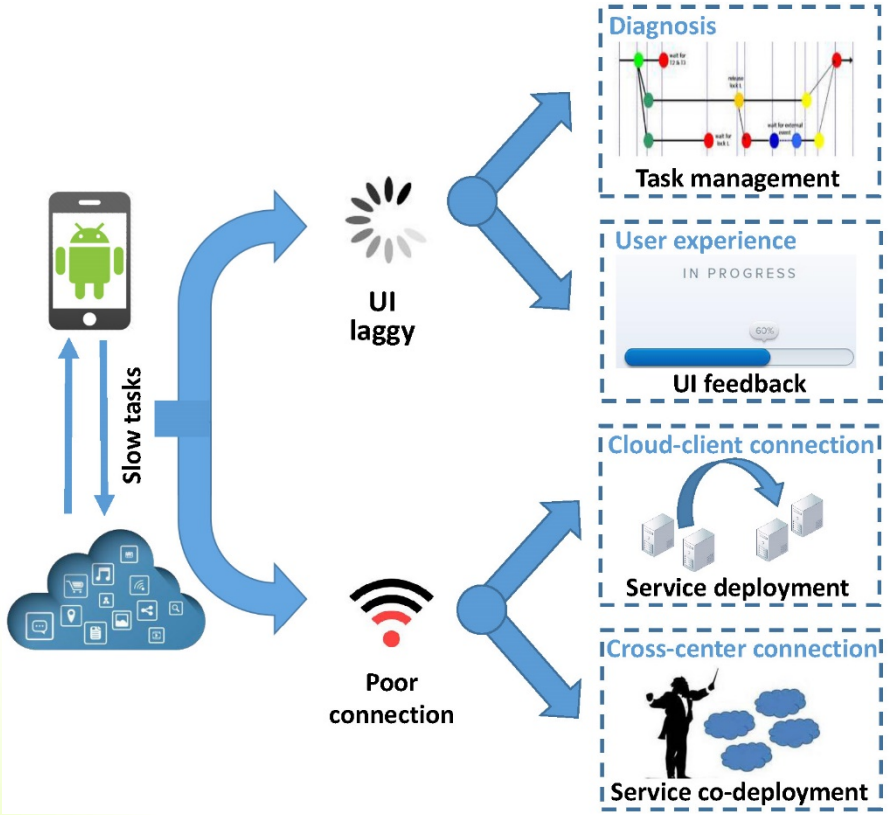


Server side performance

Client side performance

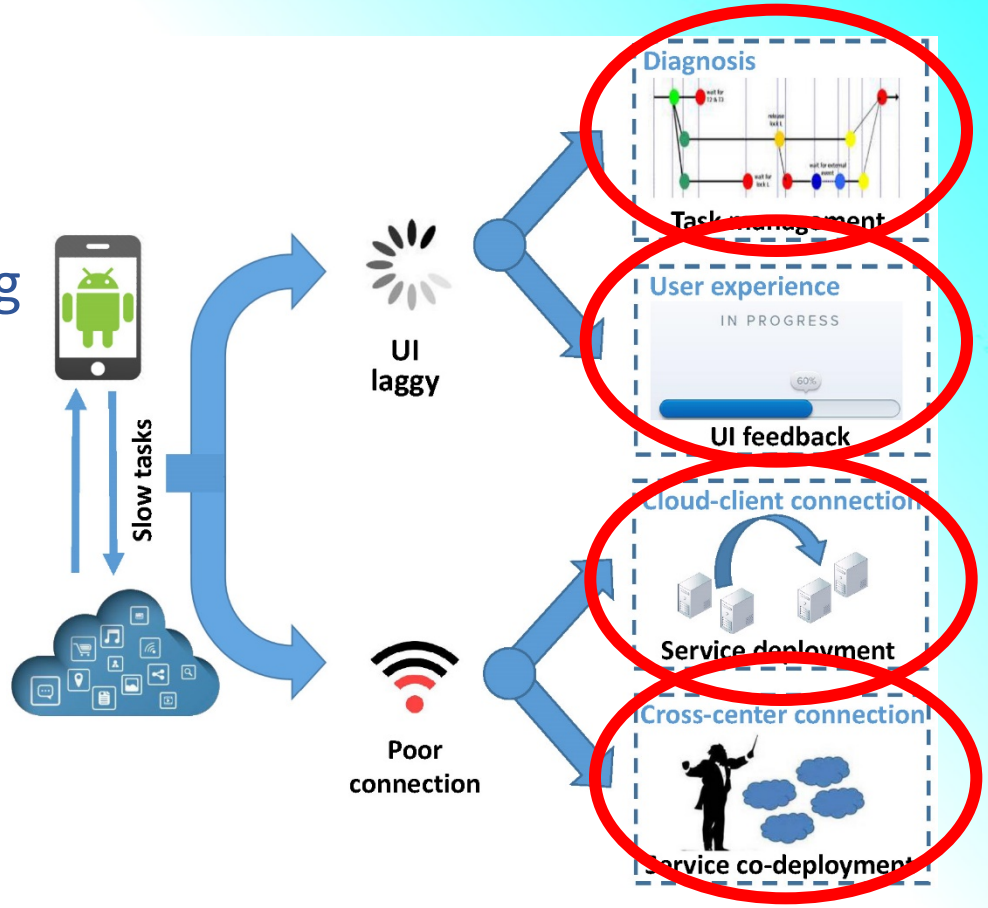
Performance Diagnosis for Cloud-Based Mobile App

- Diagnose performance on both client side and cloud side



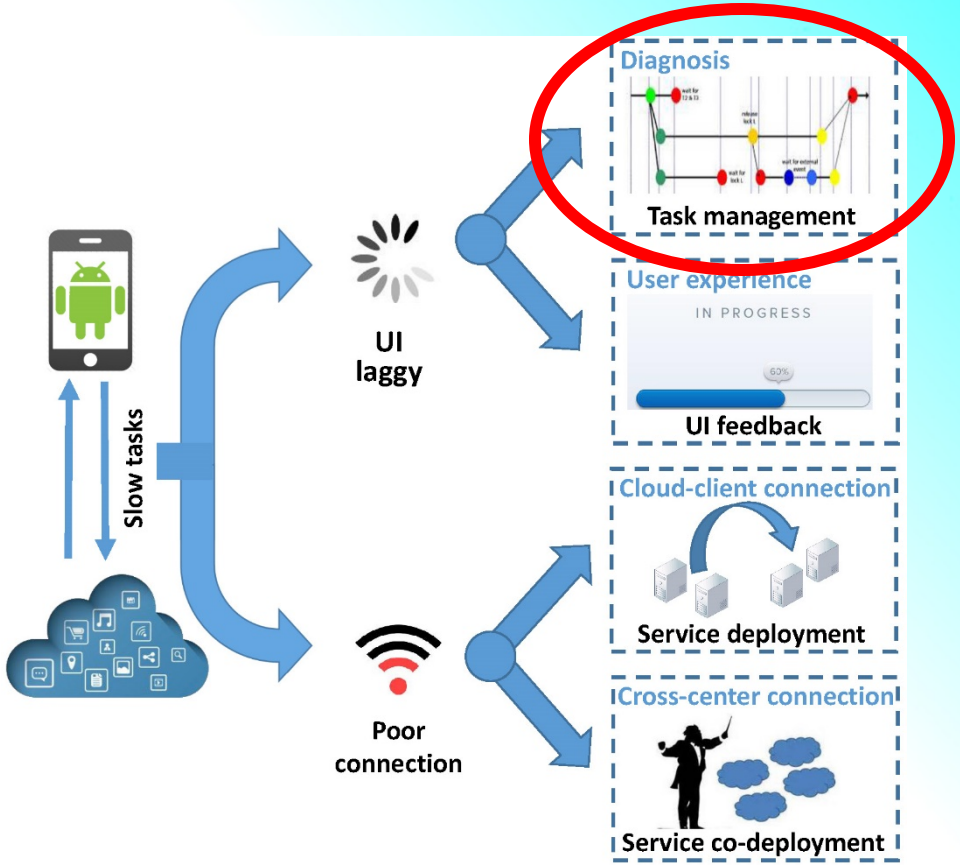
Performance Diagnosis for Cloud-Based Mobile App

- 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

	Incl Cpu Time %	Incl Cpu Time	Excl Cpu Time %	Excl Cpu Time	Incl Real Time %	Incl Real Time	Excl Real Time %	Excl Real Time	Calls+RecurCalls/Total	Cpu Time/Call ▲	Real Time/C...
entertainment/activity/AppGridFragment$AppTypeThread.r	58.0%	1156.733	0.0%	0.358	46.6%	5898.600	0.0%	0.352	6+0	192.789	983.100
entertainment/service/impl/NetServiceImpl.getAppList (II)L	58.0%	1154.926	0.0%	0.152	46.6%	5896.798	0.0%	0.154	6+0	192.488	982.800
ncurrent/ThreadPoolExecutor$Worker.run ()V	30.6%	609.927	0.0%	0.015	28.3%	3581.186	0.0%	0.016	4+0	152.482	895.297
ncurrent/ThreadPoolExecutor.runWorker (Ljava/util/concur	30.6%	609.912	0.0%	0.129	28.3%	3581.170	0.0%	0.134	4+0	152.478	895.293
entertainment/utils/HttpUtils.httpGet (Ljava/lang/String;)Lja	47.0%	937.574	0.0%	0.368	39.1%	4944.773	0.0%	0.305	6+0	156.262	824.129
iew/ViewRootImpl.performTraversals ()V	9.7%	194.197	0.0%	0.250	5.6%	706.468	0.0%	0.251	1+0	194.197	706.468
aInputStream.readLine ()Ljava/lang/String;	31.0%	617.751	9.9%	197.652	25.8%	3268.852	1.6%	197.605	6+0	102.959	544.809
/entertainment/activity/AppGridFragment$5.run ()V	13.2%	262.800	0.0%	0.540	24.2%	3060.642	0.0%	0.590	6+0	43.800	510.107
/entertainment/service/impl/DbServiceImpl.getAppType (I)	12.9%	256.840	0.0%	0.045	23.7%	3004.887	0.0%	0.049	6+0	42.807	500.815
/entertainment/dao/impl/AppTypeDaoImpl.getAppType (I)	12.9%	256.795	0.0%	0.857	23.7%	3004.838	0.0%	0.922	6+0	42.799	500.806
concurrent/locks/AbstractQueuedSynchronizer.doAcquire	0.1%	1.673	0.0%	0.251	16.1%	2033.124	0.0%	0.251	5+0	0.335	406.625
oid/internal/policy/impl/PhoneWindow$DecorView.onMea	6.6%	132.082	0.0%	0.034	3.0%	376.799	0.0%	0.033	1+0	132.082	376.799
idget/LinearLayout.onMeasure (II)V	6.6%	131.571	0.0%	0.010	2.9%	362.295	0.0%	0.009	1+0	131.571	362.295
idget/LinearLayout.measureVertical (II)V	6.6%	131.561	0.0%	0.237	2.9%	362.286	0.0%	0.236	1+0	131.561	362.286
hread.run ()V	30.6%	609.962	0.0%	0.041	28.3%	3581.221	0.0%	0.041	8+2	60.996	358.122

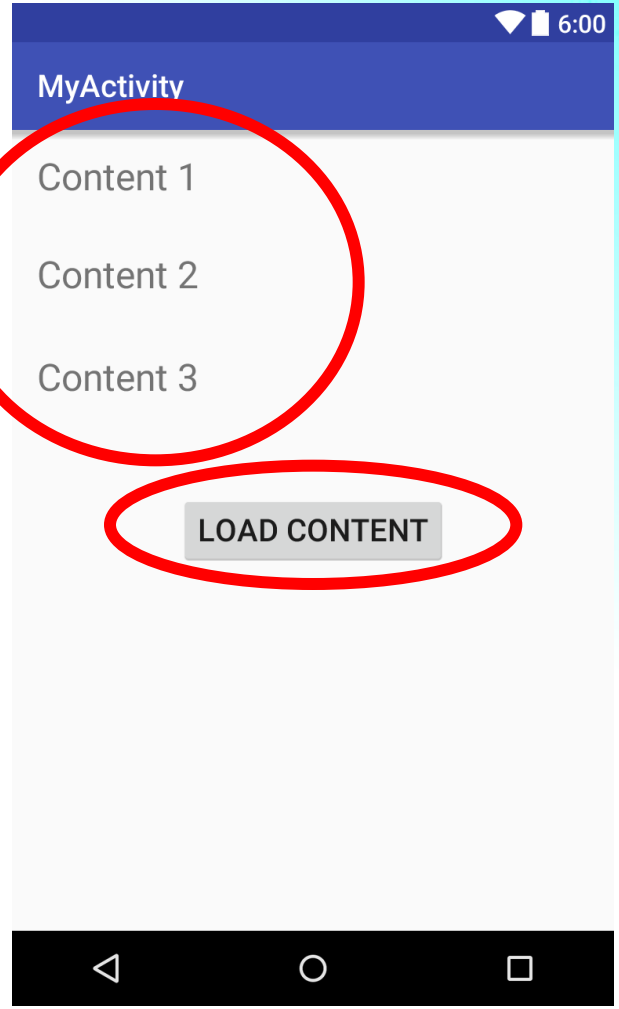
Android Application Specifics

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

- Asynchronous executions $\propto$ User perceived latency
  - Time-consuming tasks
  - Worker threads
  - Update UI afterwards

Performance Issue: an Example

```
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

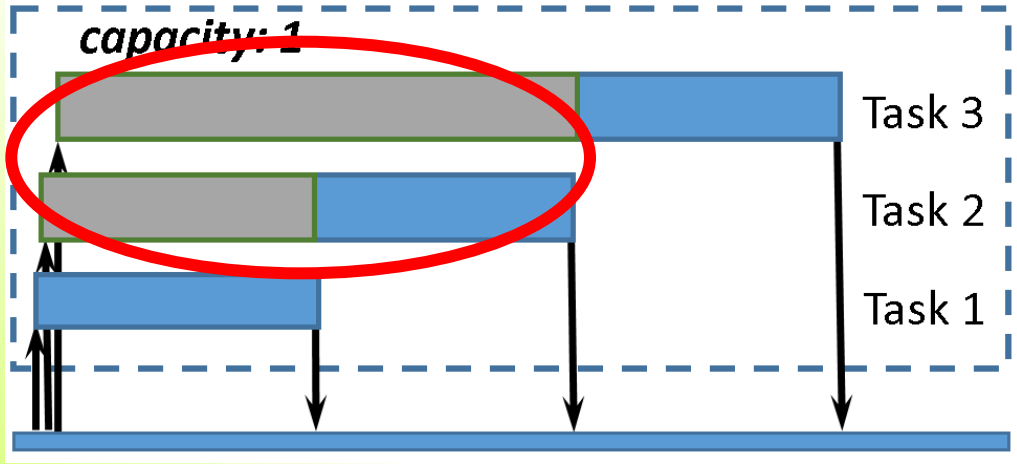
- Goal: load contents in parallel
  - Straight-forward approach
    - Call default execute method 3 times
    - Wrong!
    - Reason: tasks in a global queue
  - Correct approach
    - Resort to a thread pool
    - Call executeOnExecutor method instead
  - Existing tool (e.g., StrictMode, Asynchronizer) 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);

}
```

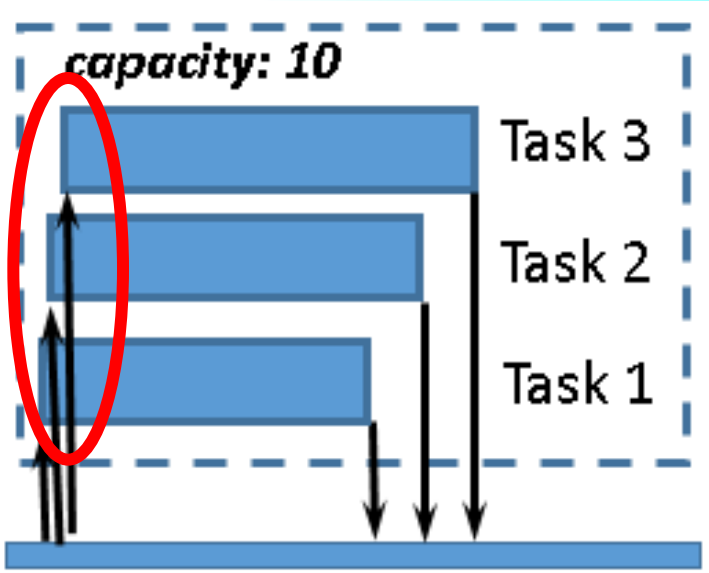
A Motivating Example in Bug Detection

Execute tasks in sequential



VS.

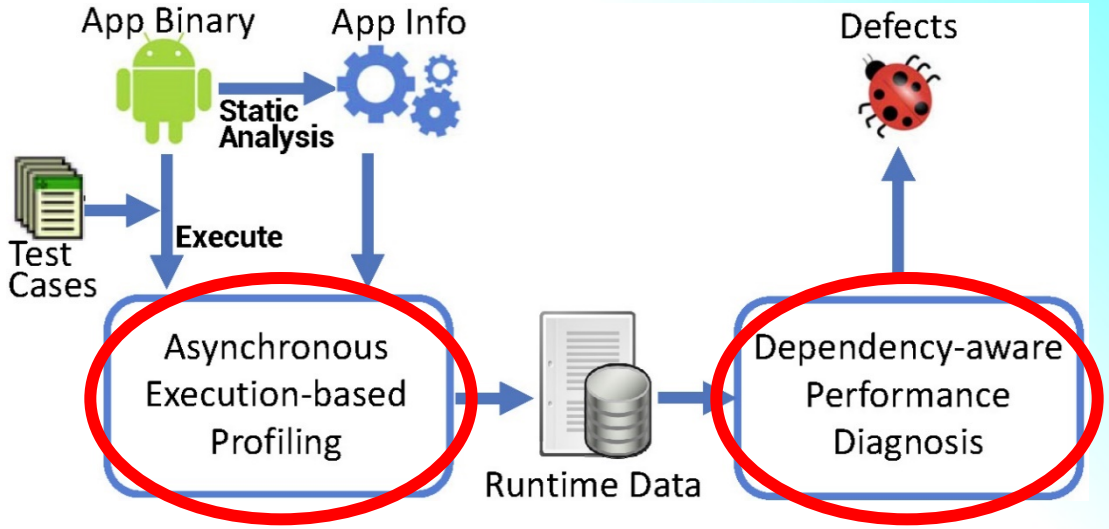
Execute tasks in parallel



- Difference: queuing time, pool capacity

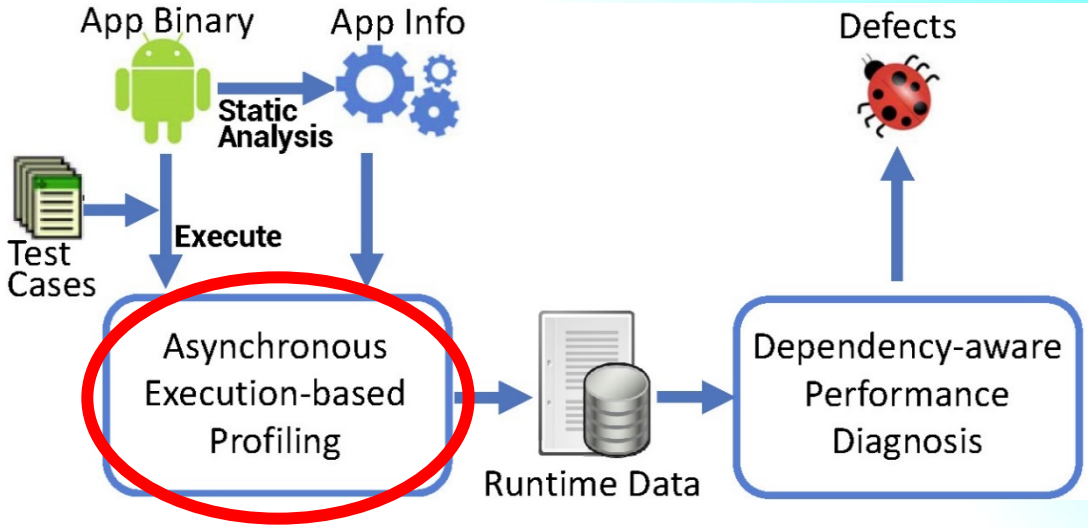
DiagDroid Framework

- Two key modules
  - Profiler
  - Log analyzer
- Two parts
  - PC & Mobile
- Two mechanisms
  - Static & Dynamic Analysis



Profiler

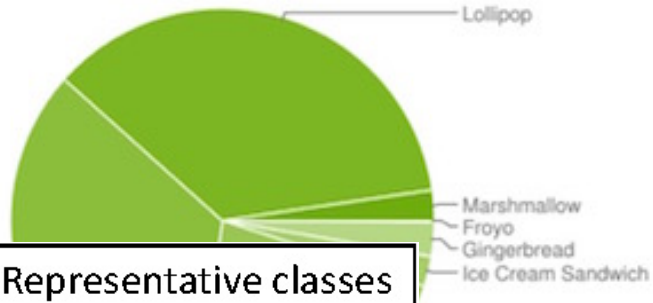
- Features to profile
  - Runtime info (e.g., Queuing time, Execution time)
  - Identification info (e.g., Execution context (call-stack), pool identifier)



Profiler

- Chall
  1.
  2.
  3.
- Solut
  1.
  2.
  3.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.6%
4.0			
4.0			
4.1			
4.2			
4.3			
4.4			
5.0	Lollipop	21	16.9%
5.1		22	19.2%
6.0	Marshmallow	23	2.3%



Category	Type	Representative classes
Reusing existing threads	Looper & Handler	HandlerThread
		IntentService
	Pool-based executor	ThreadPoolExecutor
		AsyncTask
Creating new threads	Thread	Thread

ethods

Log Analyzer

1461 → 75

```
java.util.concurrent.ThreadPoolExecutor
java.util.concurrent.ThreadPoolExecutor.execute(Native Method)
java.util.concurrent.Executors$DelegatedExecutorService.execute(Executors.java:552)
android.app.SharedPreferencesImpl.enqueueDiskWrite(SharedPreferencesImpl.java:539)
android.app.SharedPreferencesImpl.access$100(SharedPreferencesImpl.java:52)
android.app.SharedPreferencesImpl$EditorImpl.apply(SharedPreferencesImpl.java:381)
android.preference.PreferenceManager.setNoCommit(PreferenceManager.java:532)
```

- Tar
- Analysis
-

	Context c_1	Context c_2	Context c_3
1.	de.jdsoft.law.data.UpdateLawList	de.jdsoft.law.data.UpdateLawList	de.jdsoft.law.data.UpdateLawList
2.	android.os.AsyncTask.executeOnExecutor(Native Method)	android.os.AsyncTask.executeOnExecutor(Native Method)	android.os.AsyncTask.executeOnExecutor(<Xposed>)
3.	android.os.AsyncTask.execute(AsyncTask.java:534)	android.os.AsyncTask.execute(AsyncTask.java:534)	android.os.AsyncTask.execute(AsyncTask.java:539)
4.	de.jdsoft.law.LawListFragment.onCreate(LawListFragment.java:91)	de.jdsoft.law.LawListFragment.onCreate(LawListFragment.java:91)	de.jdsoft.law.LawListFragment.onCreate(LawListFragment.java:91)
	…	…	…
9.	android.support.v4.app.FragmentActivity.onCreateView(FragmentActivity.java:285)	android.view.LayoutInflater.createViewFromTag(LayoutInflater.java:676)	android.view.LayoutInflater.createViewFromTag(LayoutInflater.java:727)
	…	…	…

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

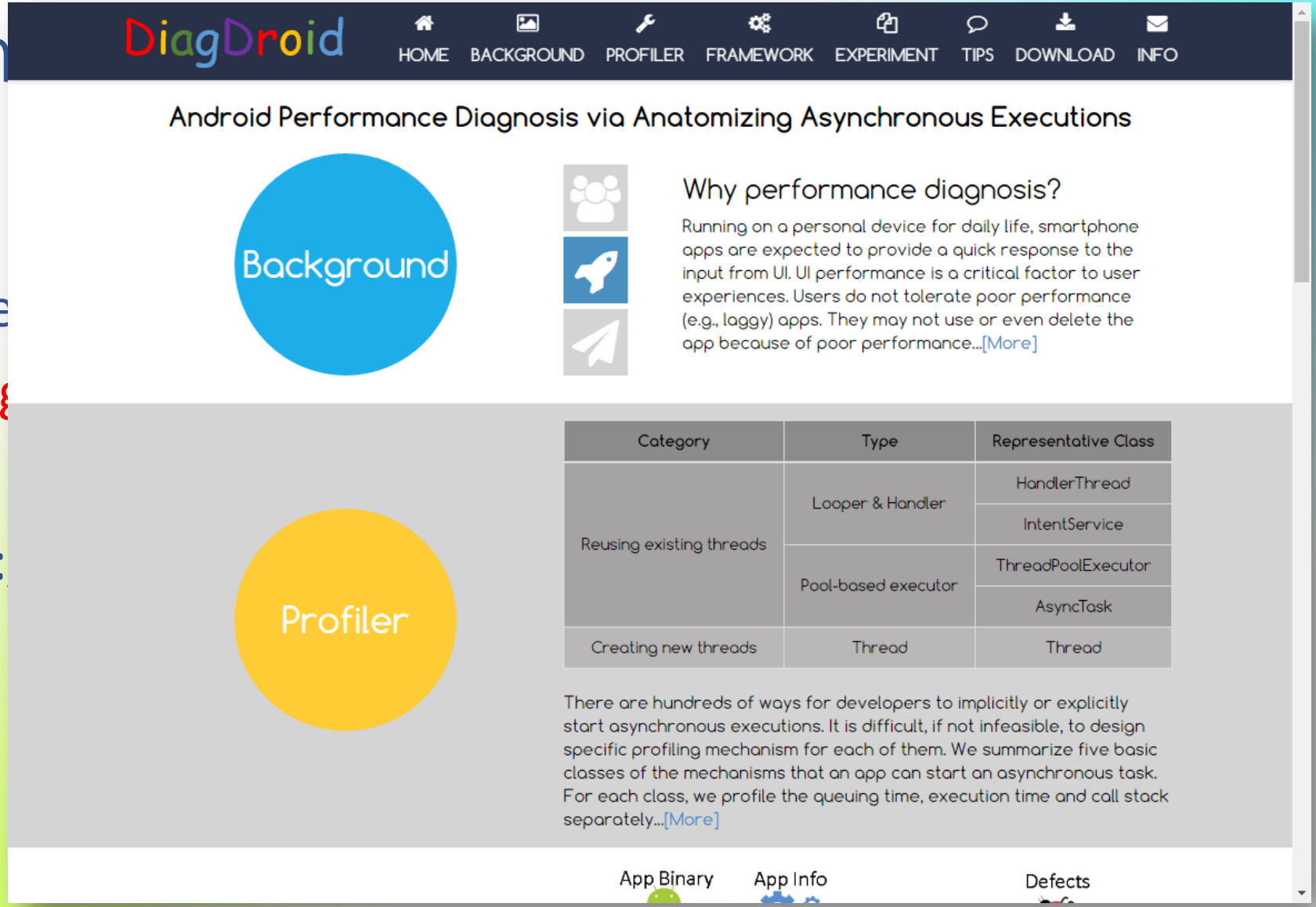
- Found: 27 new bugs of 5 types in 33 open source apps
  1. Sequential execution
  2. Forgetting cancelling execution
  3. Improper execution pool
  4. Message queue overloading
  5. Misusing third-party library
- Note:
  - No a priori knowledge on the app

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

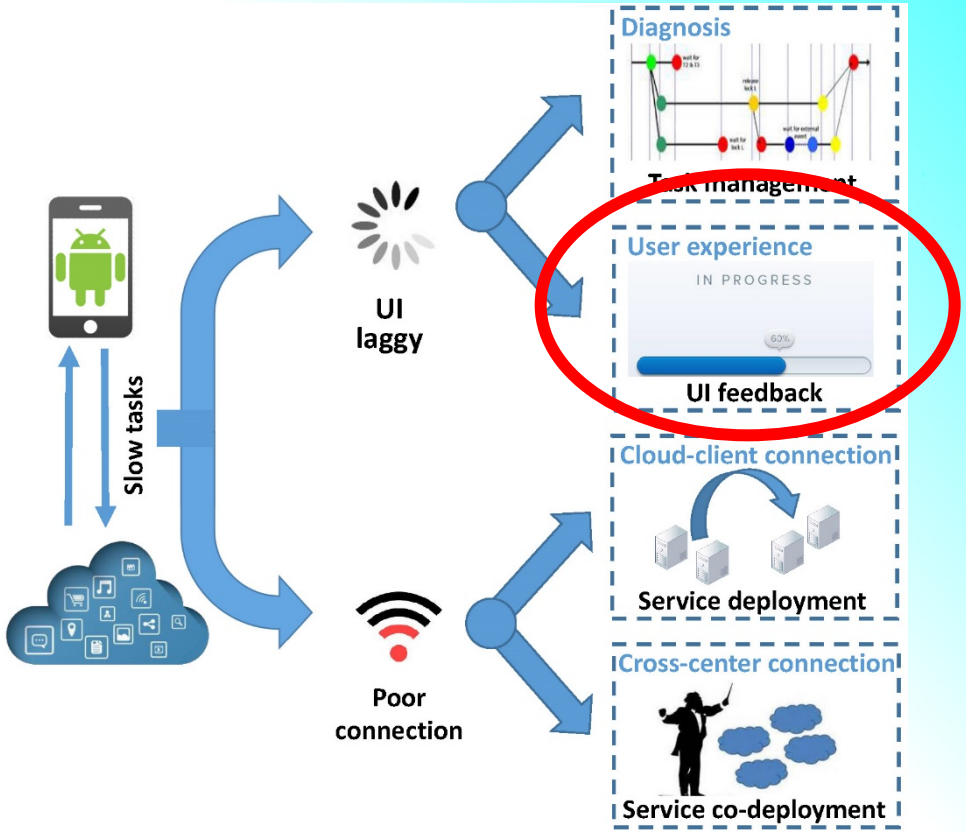
Sum

- New
- Nove
- Categ
- New
- http:



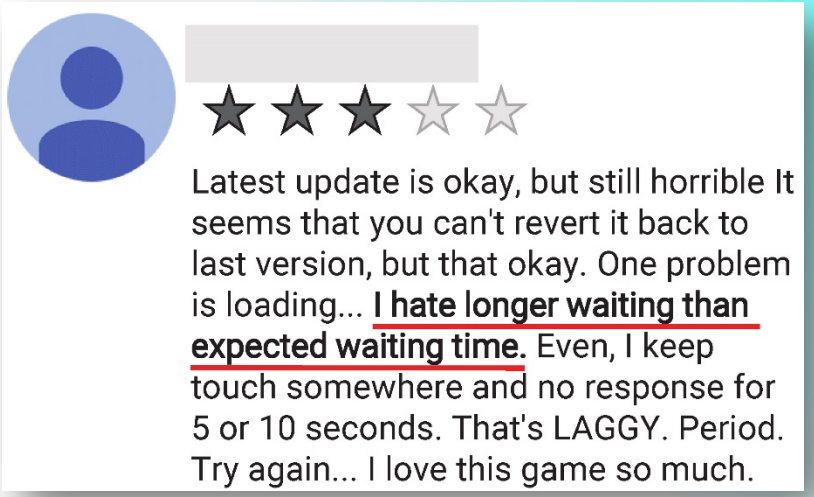
Outline

- 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



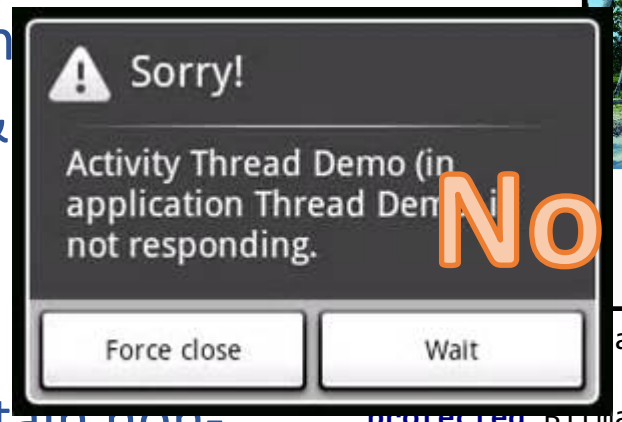
Introduction

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



Motivation

- Android unique UI mechanism
  - Activity not respondin
  - Asynchronous tasks &
- UI responsiveness
  - Simple code may contain non-responsive UI design



No Feedback

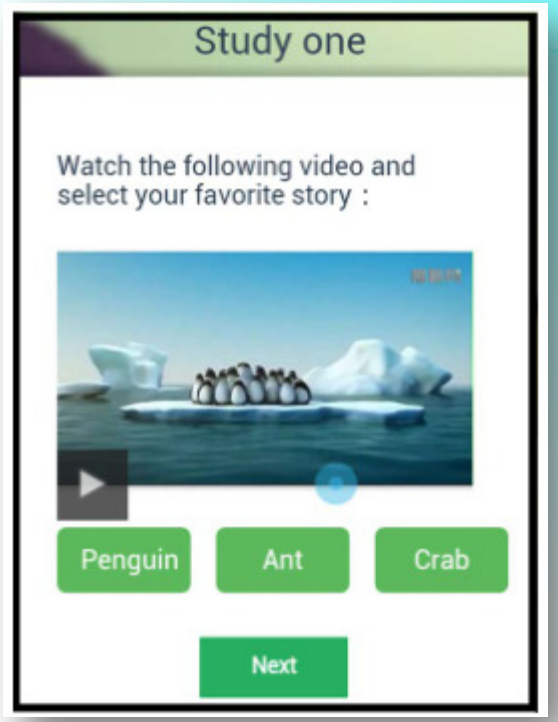
```
ageDownloader extends AsyncTask<String, Void,

  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

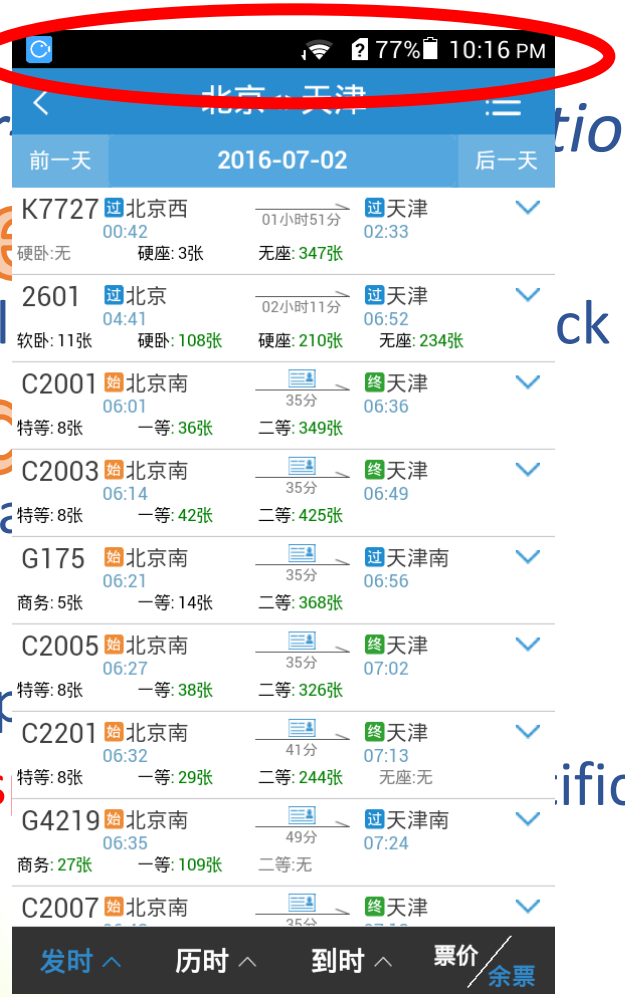


Problem Specification

- 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 ≥ T (T=500ms)

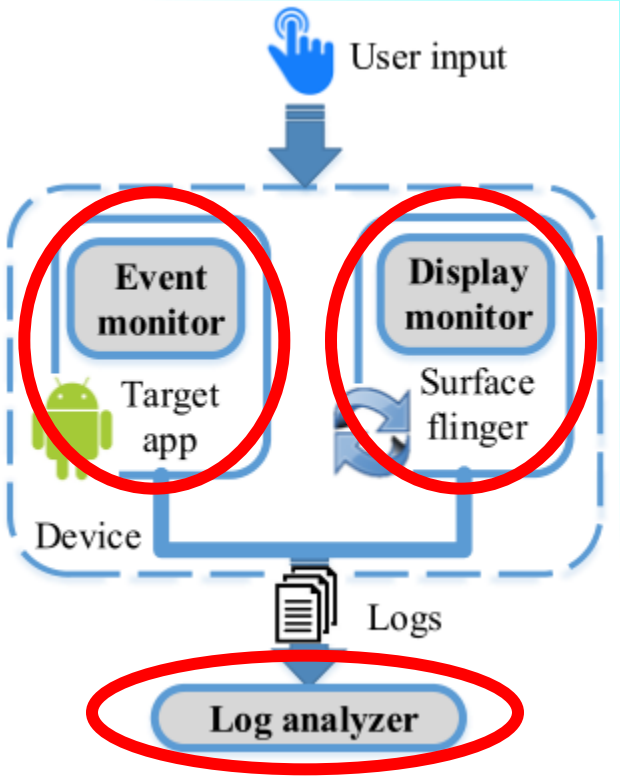
Challenging Task

- Task: detecting *poor* *tions*
- Current tool Faile
  - Detecting abnormal ck delay (e.g., DiagDroid)
- First Trial Failed
  - Monitoring all displa
- Solution
  - Monitor UI update p
  - Separate system dis ification bar) with app UI updates



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_artis
tic}_null-Motion-UP : 125696
```

Implementation

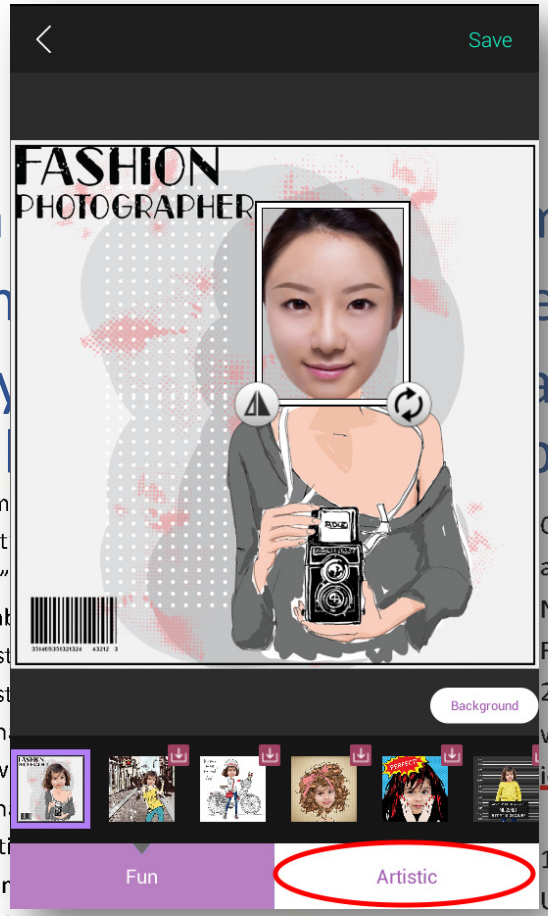
- Display monitor
  - Hook C inter-process communication (IPC) to surfaceflinger
  - Sample log:

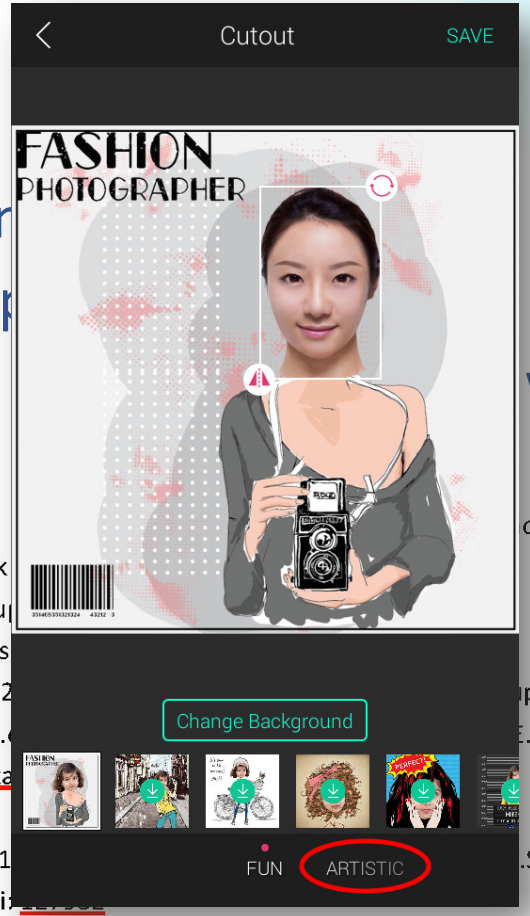
```
... 138 ...: BIPC:****android.gui.SurfaceTexture****, sender_pid:2365, UptimeMilli: 127932
```

- 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
  - Popula
  - Confirn
  - “With y
    We will





nillior
evelop
at we
ble.”
which tends to be slower.

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

Operation:
artistic Click
No screen u
Related logs
2365
widgetpool.
id/cutout_ta
138
UptimeMilli

on.ChildProportionLayout cutout_tab_
perfect[Event]com.cyberlink.youperfect.
...C....P....270,0-540,67#7f0a051dapp:
.SurfaceTexture****, sender_pid:2365,

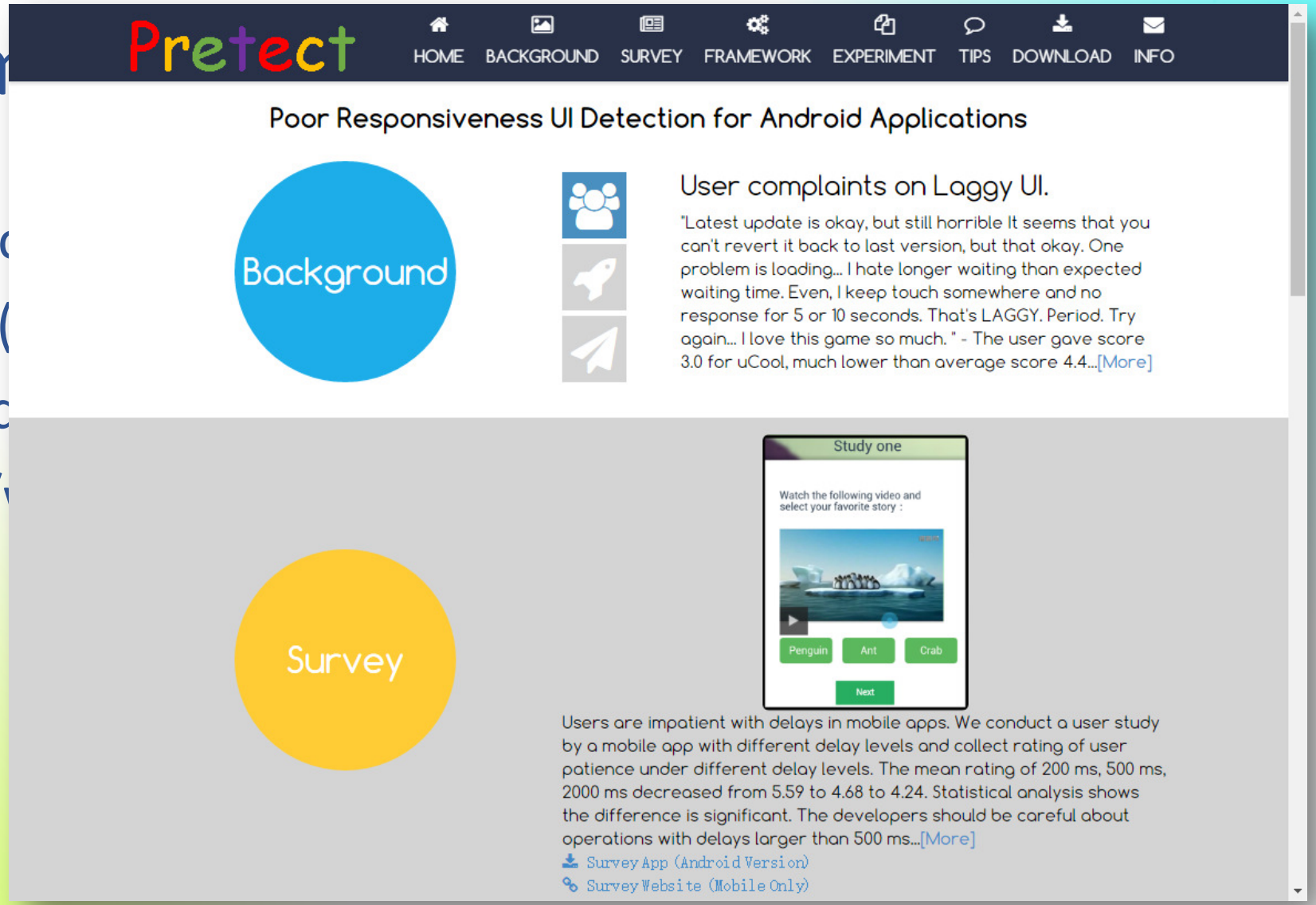
YouCam - steps to reproduce

Ver. 4.10 vs. Ver. 5.4

YouCam – sample reports

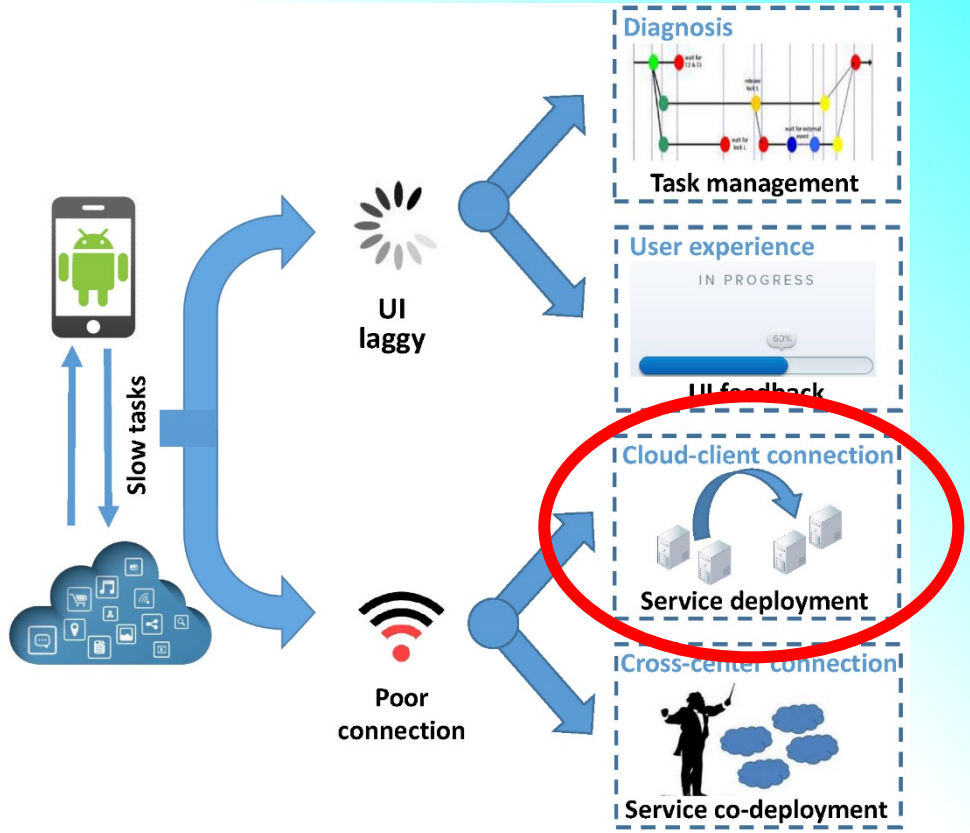
Summ

- Real wo
- A tool (
- Cases c
- http://



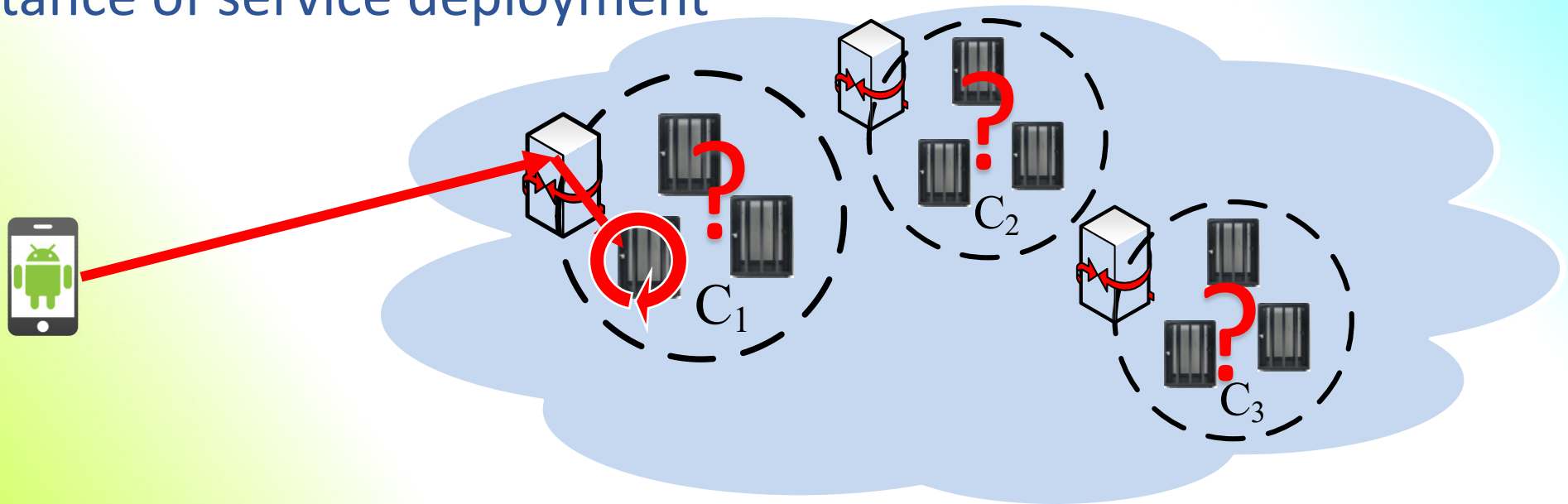
Outline

- Introduction
- Client Side Performance Diagnosis
- Cloud 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

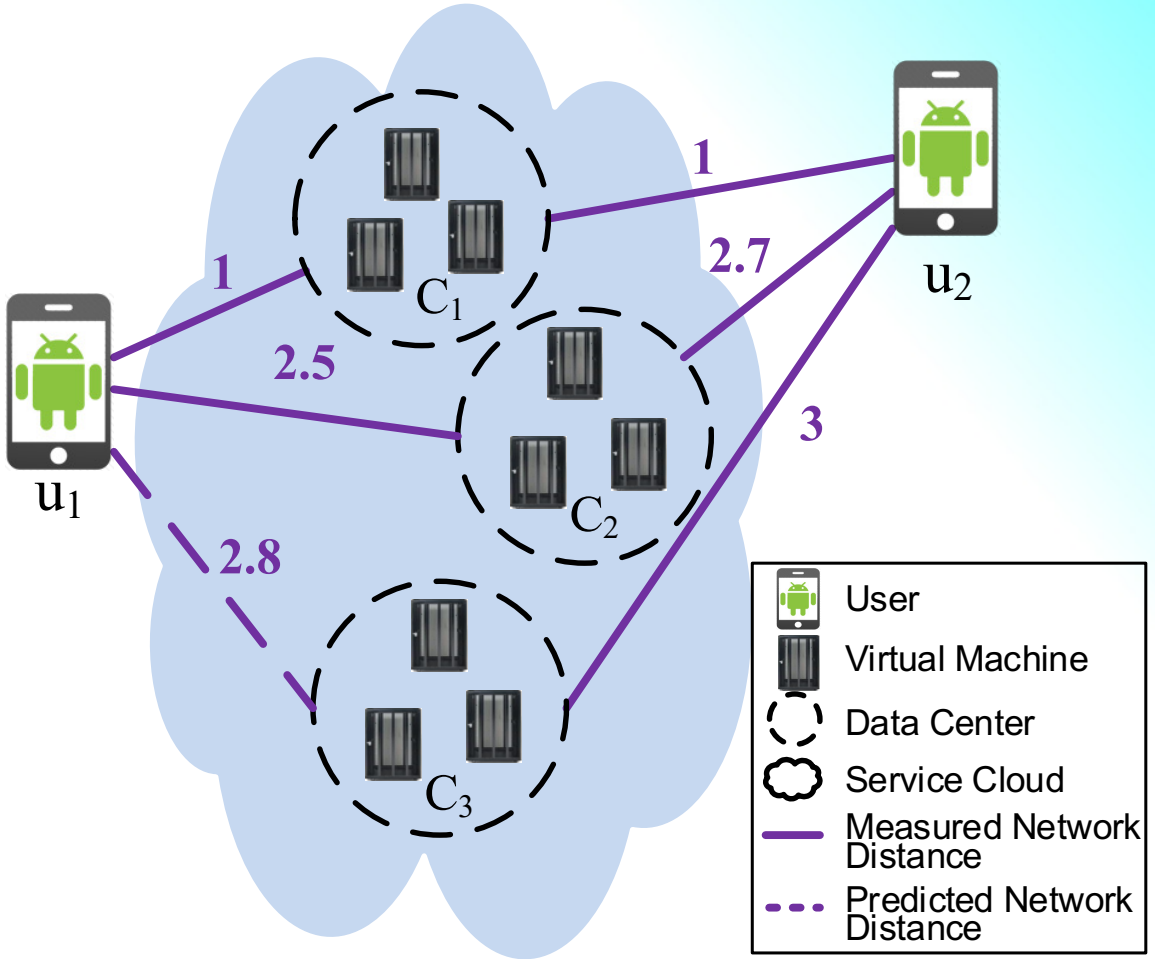


Optimizing UC Delay

- 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

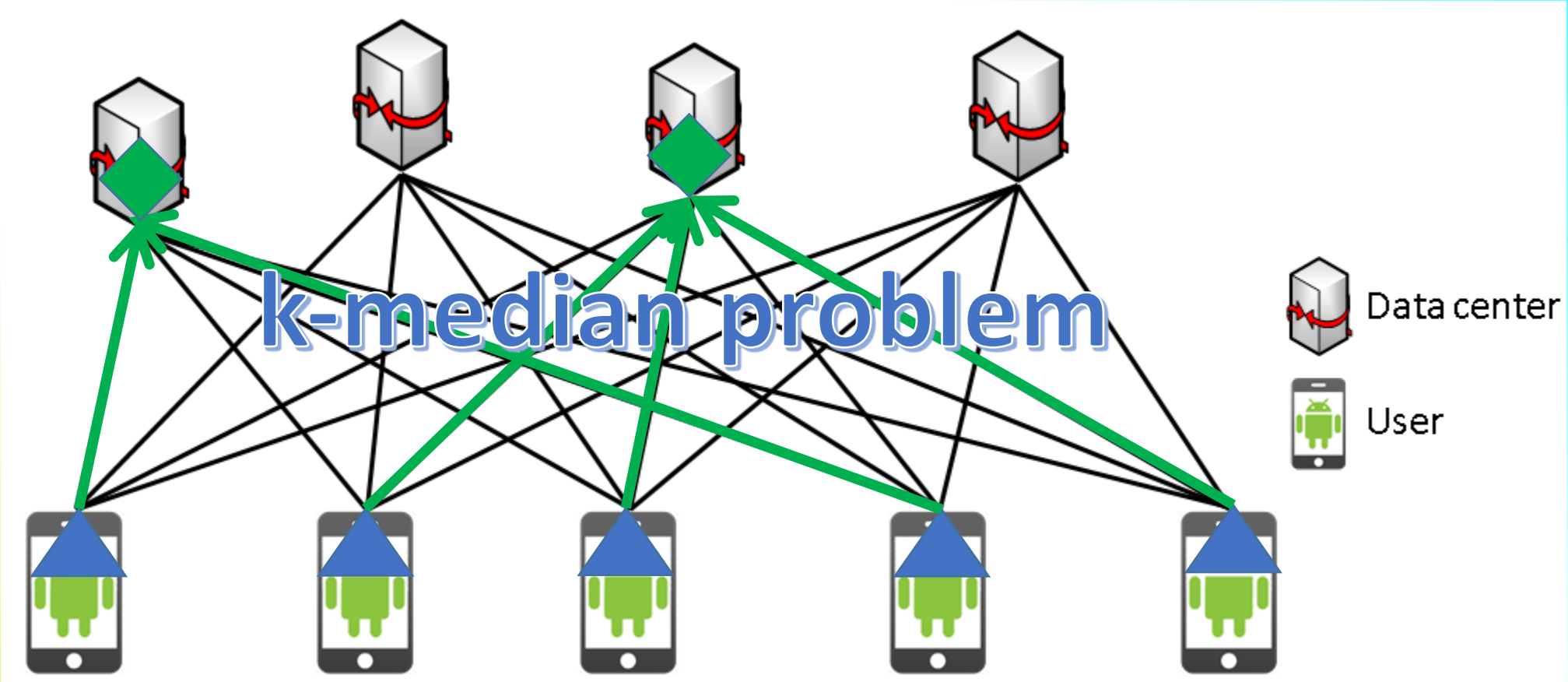
Minimize:

$$\sum_{i=1}^{N} \min_{j \in Z'} \{d_{ij}\}$$

Subject to:

$$Z' \subset Z, |Z'| = k$$

Minimize Average Cost



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

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

$i \in V_1, j \in V_2$

$\{\blacksquare(i,j)\in E, d\downarrow ij \leq T \ (i,j)\notin E, otherwise.$

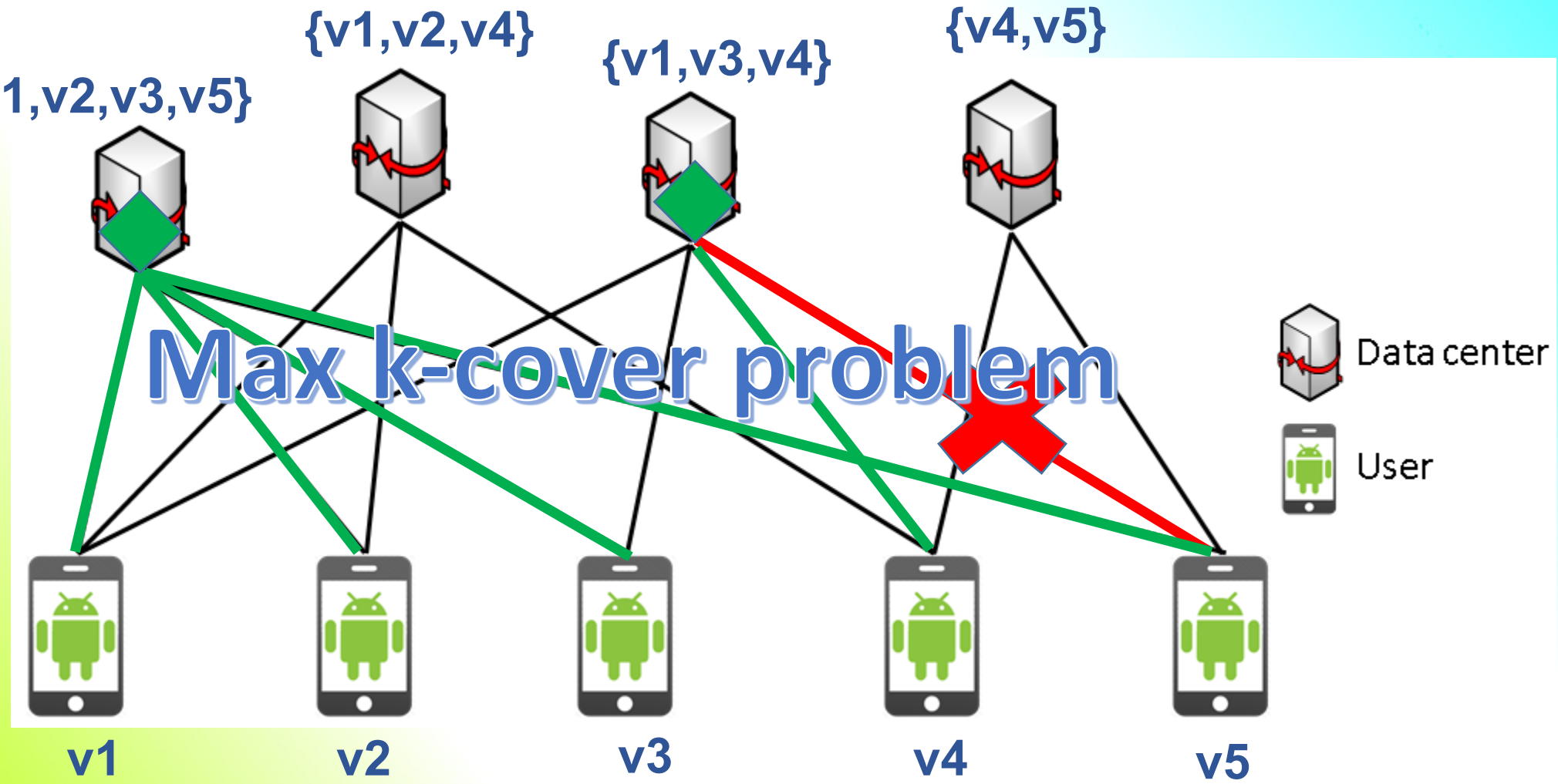
Maximize:

$$|N_B(V')|$$

Subject to:

$$V' \subset V_1, \ |V'| = k$$

Maximize Close User Amount

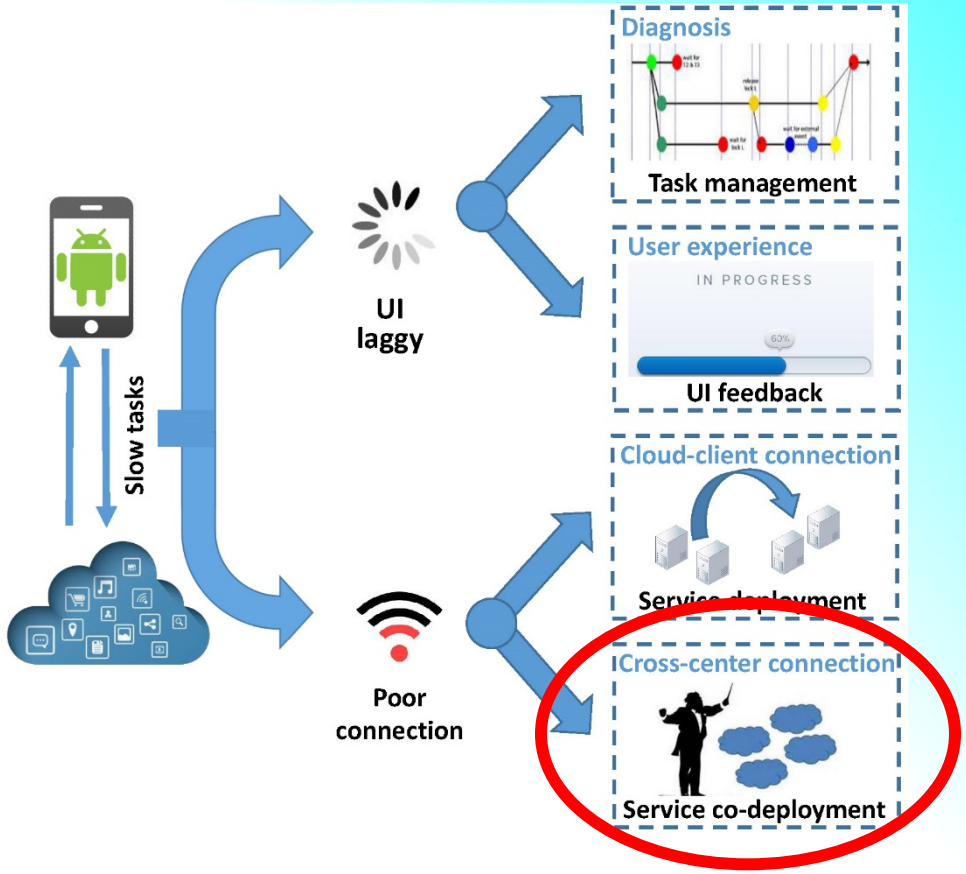


Real-World Dataset

- 303 PlanetLab computers
- 4,302 the Internet services
- ≈130,000 response-time values matrix

Outline

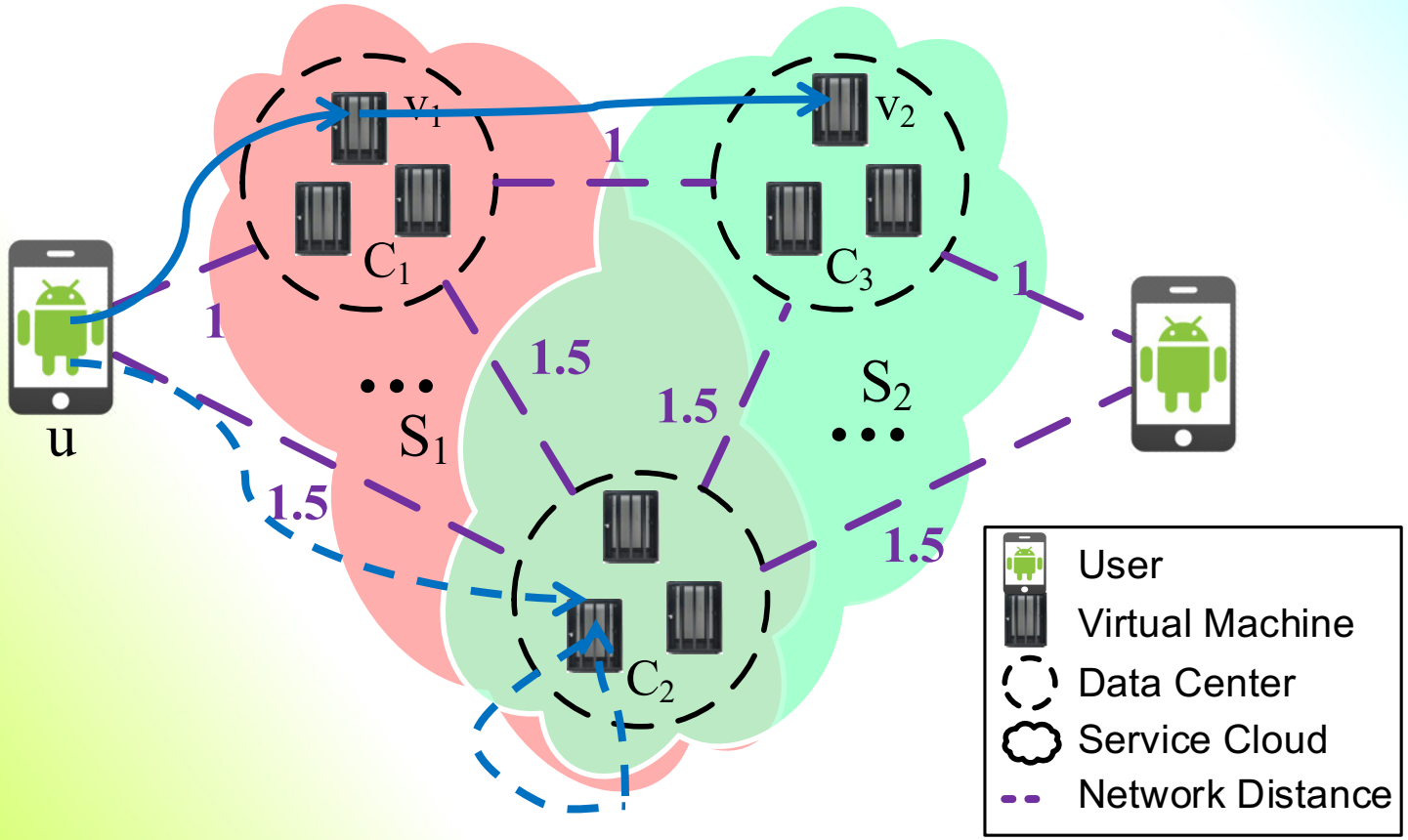
- Introduction
- Client Side Performance Diagnosis
- Cloud Side Performance Diagnosis
  - Deployment of Single Cloud Service
  - Deployment of Multiple Cloud Services
- Conclusion



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

$$\text{minimize} \sum_{\substack{i\in U\\ 1\le h\le m\\ j\in C_h}} r_{hi} d_{hij} x_{hij} + \sum_{\substack{i\in U\\ 1\le q,s\le m, q\ne s\\ p\in C_q, r\in C_s}} r_{iqs} d_{pqrs} y_{ipqrs}$$

subject to:

$$\sum_{j\in C_h} x_{hij} = sign(r_{hi}), \qquad 1 \le h \le m, \forall i \in U,$$

$$x_{hij} \le z_{h}, \qquad 1 \le h \le m, \forall j \in C_h, \forall i \in U,$$

$$\sum_{j\in C_k} z_{hj} \le k_h, \qquad 1 \le h \le m,$$

$$\sum_{\substack{1\le s\le m\\ s\ne h\\ r\in C_s}} y_{ijhrs} \le x_{hij}, \qquad 1 \le h \le m, \forall j \in C_h, \forall i \in U,$$

$$y_{ipqrs} \le z_{rs}, \qquad 1 \le q, s \le m, q \ne s, \forall p \in C_q, \forall r \in C_s, \forall i \in U,$$

$$\sum_{\substack{p\in C_q\\ r\in C_s}} y_{ipqrs} = sign(r_{iqs}), \qquad 1 \le q, s \le m, q \ne s, \forall i \in U,$$

$$x_{hij} \in \{0,1\}, \qquad 1 \le h \le m, j \le C_h, \forall i \in U,$$

$$y_{ipqrs} \in \{0,1\}, \qquad 1 \le q, s \le m, q \ne s, \forall p \in C_q, \forall r \in C_s,$$

$$z_{hj} \in \{0,1\}, \qquad 1 \le h \le m, \forall j \in C_h.$$

Integer programming

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

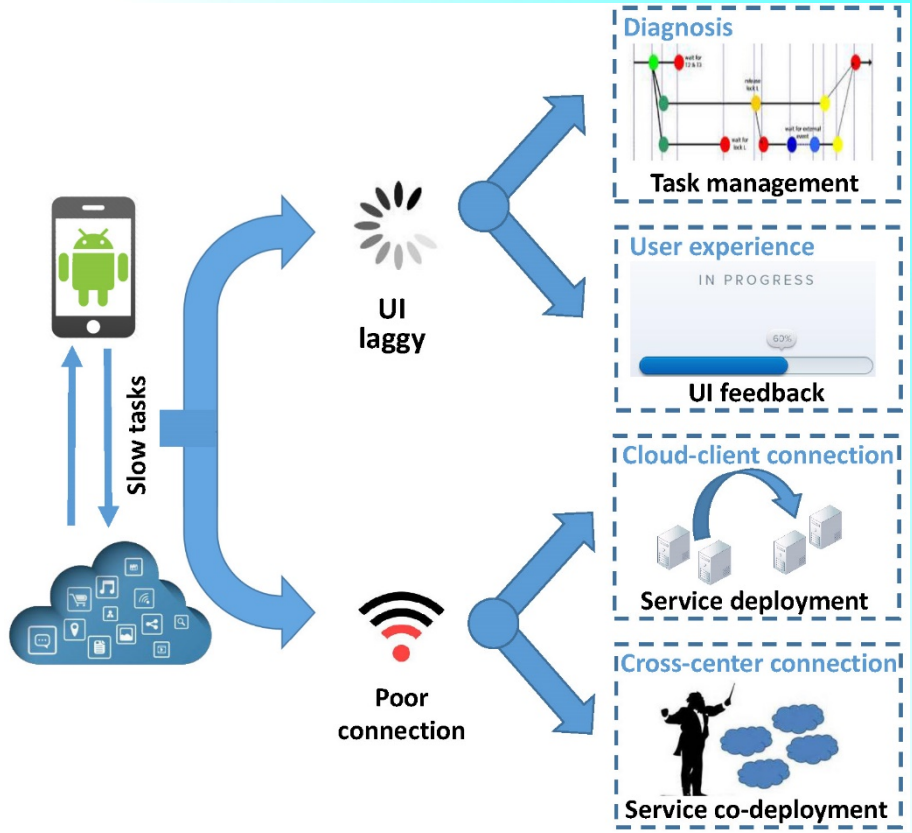
- Model user experience
- Formulate deployment problems
- Real-world dataset
- http://appsrv.cse.cuhk.edu.hk/~ykang/cloud

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

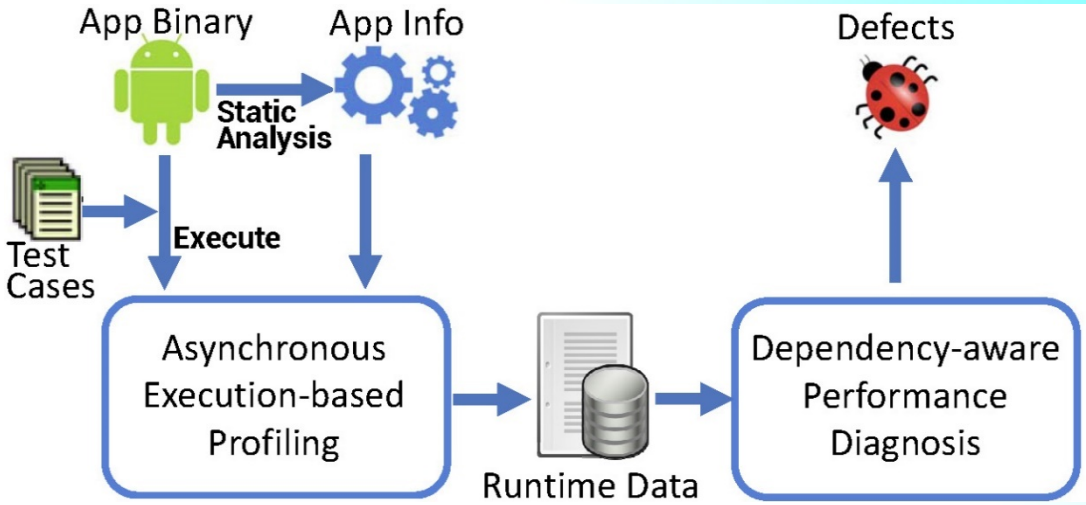
Performance issue 2 – correct code

```
public class MyActivity extends Activity {

 private class RetrieveInfoTask extends
AsyncTask<String, Void, String> {

  ...

  @Override

  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

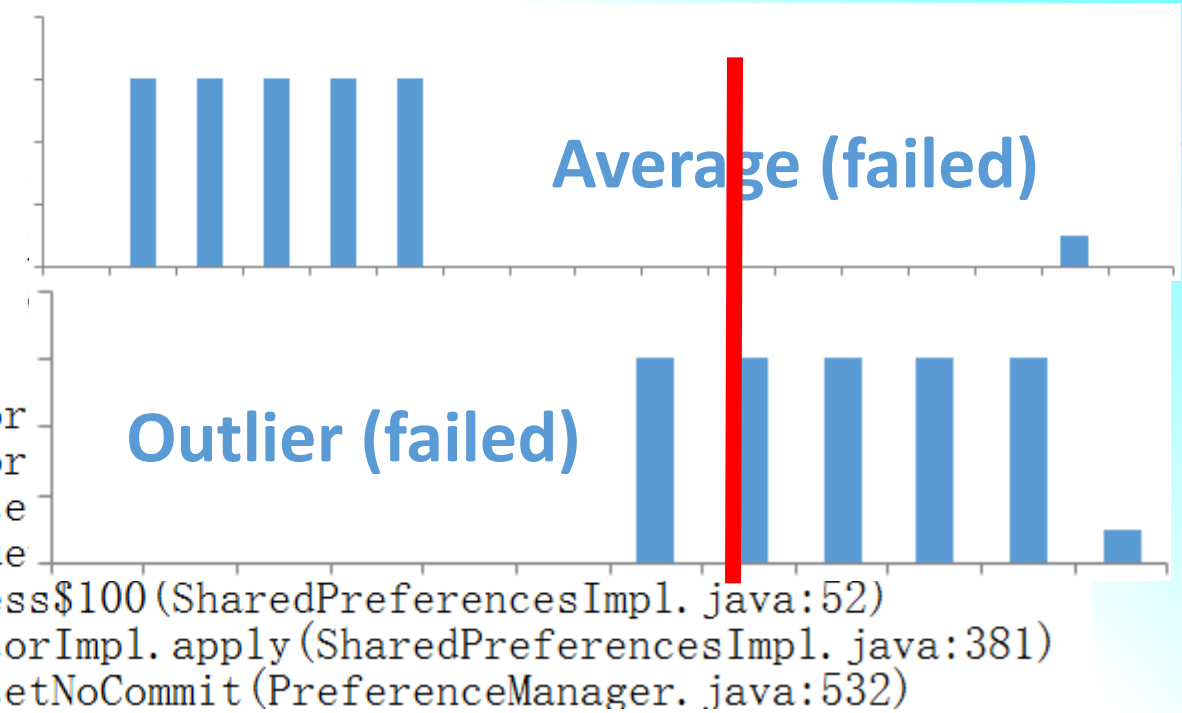


Profiling mechanisms (example)

- 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)
  - 1461 → 75
- Sol
  - java.util.concurrent.ThreadPoolExecutor
  - java.util.concurrent.ThreadPoolExecutor
  - java.util.concurrent.Executors$Delegate
  - android.app.SharedPreferencesImpl.enque
  - android.app.SharedPreferencesImpl.access$100(SharedPreferencesImpl.java:52)
  - android.app.SharedPreferencesImpl$EditorImpl.apply(SharedPreferencesImpl.java:381)
  - android.preference.PreferenceManager.setNoCommit(PreferenceManager.java:532)



Other modules

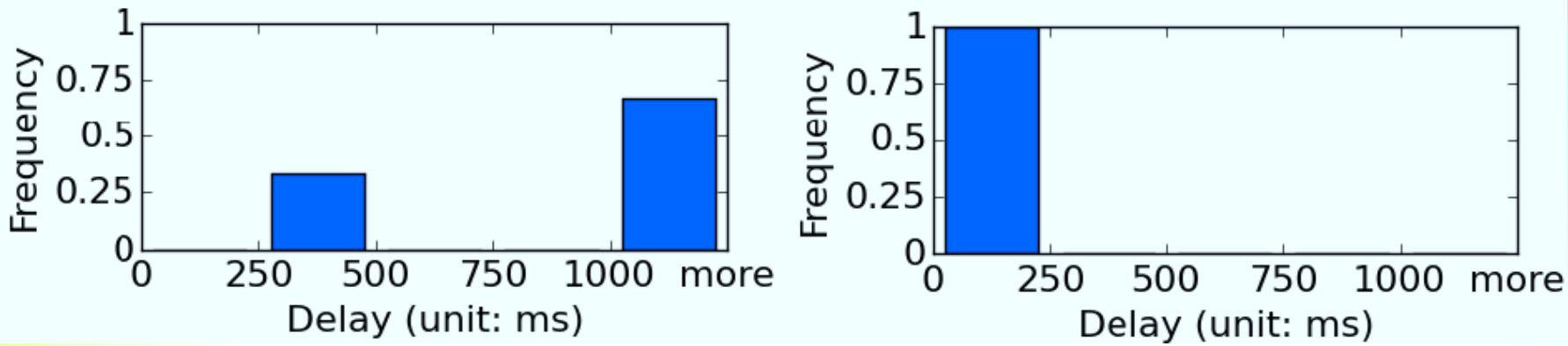
- Static analysis
  - Decompile the app and gather information from bytecode
  - Do not modify the original app
- Test executor
  - A guard of Monkey Exerciser (a random testing tool)
  - Support plugin of any kind of test scripts

DiagDroid - Bugs found

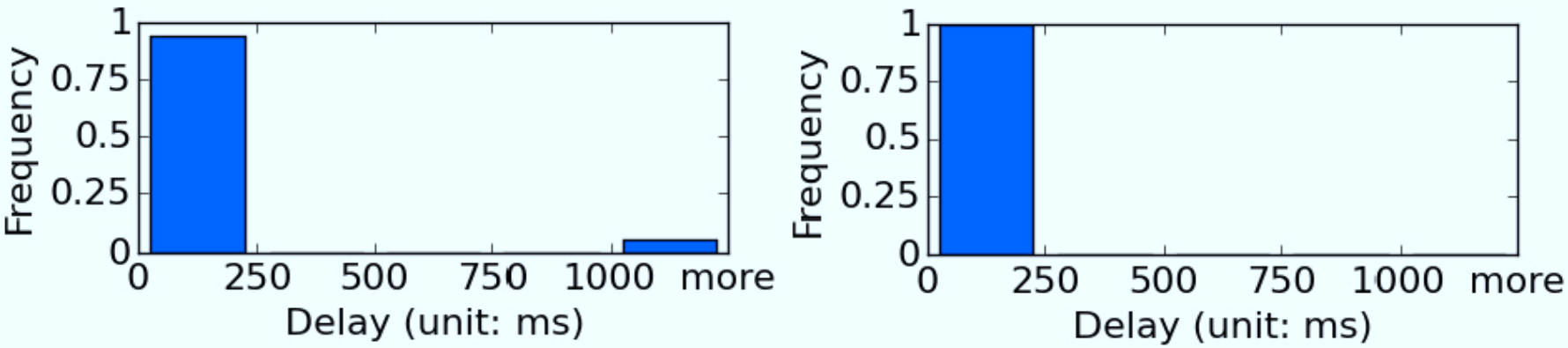
- www.cudroid.com/DiagDroid

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

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.
4. Keep effective response.
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

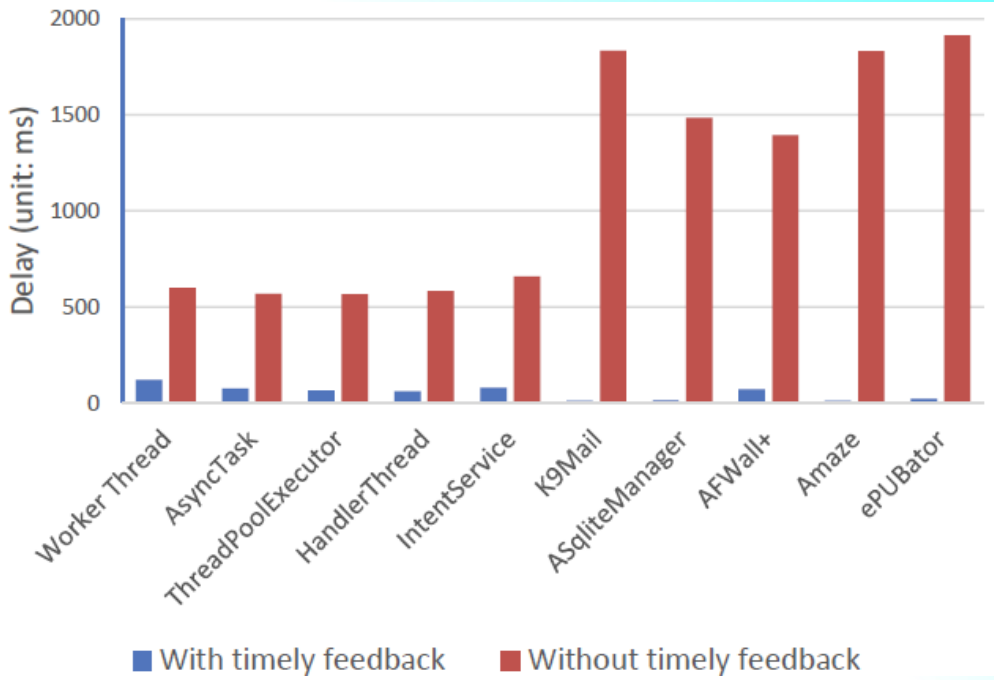
Delay level	Mean	Std. Deviation	N
200 ms	5.59	2.14	38
500 ms	4.68	1.80	37
2000 ms	4.24	2.43	41
Total	4.82	2.21	116

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

delay_level Simple Contrast			Dependent Variable B2Items	delay_level Simple Contrast			Dependent Variable B2Items
500 ms vs. 200 ms	Contrast Estimate		-0.92	2000 ms vs. 200 ms	Contrast Estimate		-1.35
	Hypothesized Value		0.00		Hypothesized Value		0.00
	Difference (Estimate -		-0.92		Difference (Estimate -		-1.35
	Std. Error		0.50		Std. Error		0.48
	Sig.		0.07		Sig.		0.01
	95% Confidence Interval for Difference	Lower Bound	-1.90		95% Confidence Interval for Difference	Lower Bound	-2.31
		Upper Bound	0.07			Upper Bound	-0.39

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



Experiment

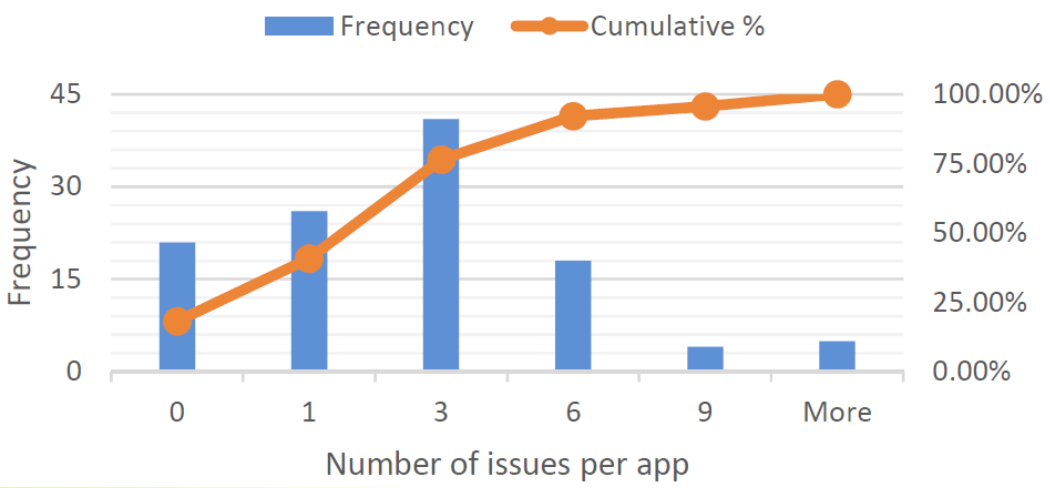
- Overall
  - Apply to 115 popular Android apps covering 23 categories (including BooksReferences, Photography, Sports, etc)

App Statistics		Issues Statistics	
# apps contain bugs	94	Total	327
Max bugs an app	23	Max	29189.0 (ms)
Min bugs an app	0	Min	504.0 (ms)
Avg. bugs per app	2.8	Avg	1603.9 (ms)
Median bugs per app	2	Stdv	2635.5

Experiment Stats

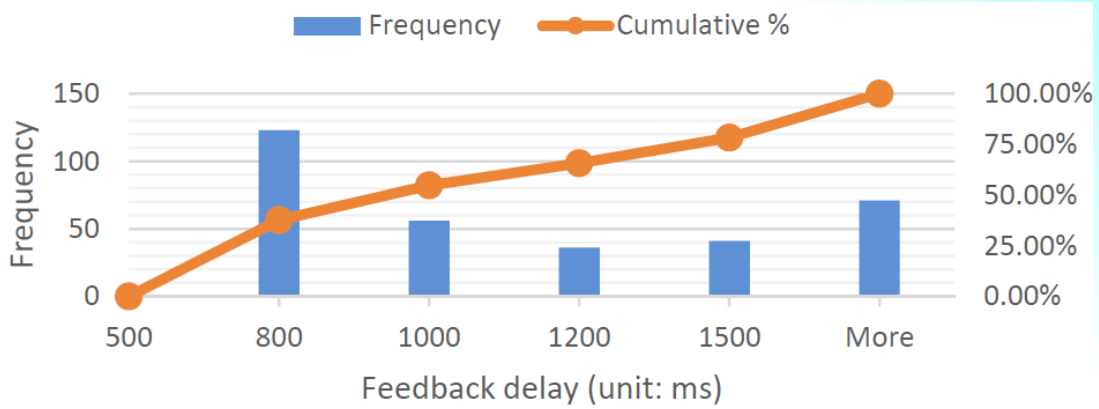
- 94/115 apps contain potential UI design defects
- 327 independent components with feedback delays ≥ 500 ms
- Maximum delay ≥ 29 s

Experiment Stats

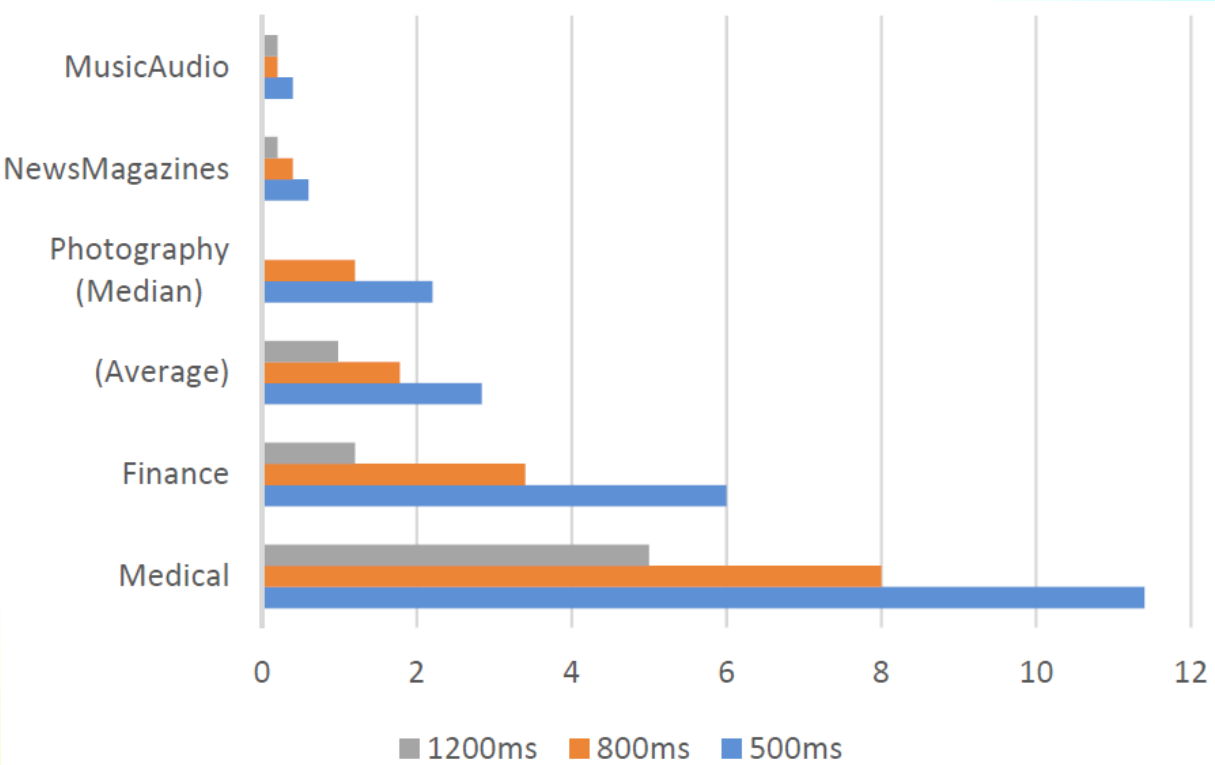


Distribution of number of issues per app

Distribution of feedback delay per app



Different threshold



Avg. number of cases per category by threshold

Poor-responsive Components

Component	Issues found
android.widget.Button	78
android.widget.ListView	30
android.widget.ImageButton	22
android.widget.EditText	21
android.widget.ScrollView	20
android.widget.RelativeLayout	16
android.widget.ImageView	13
android.widget.TextView	10
android.widget.LinearLayout	10
android.support.v7.widget.Toolbar	7

Single Service Deployment

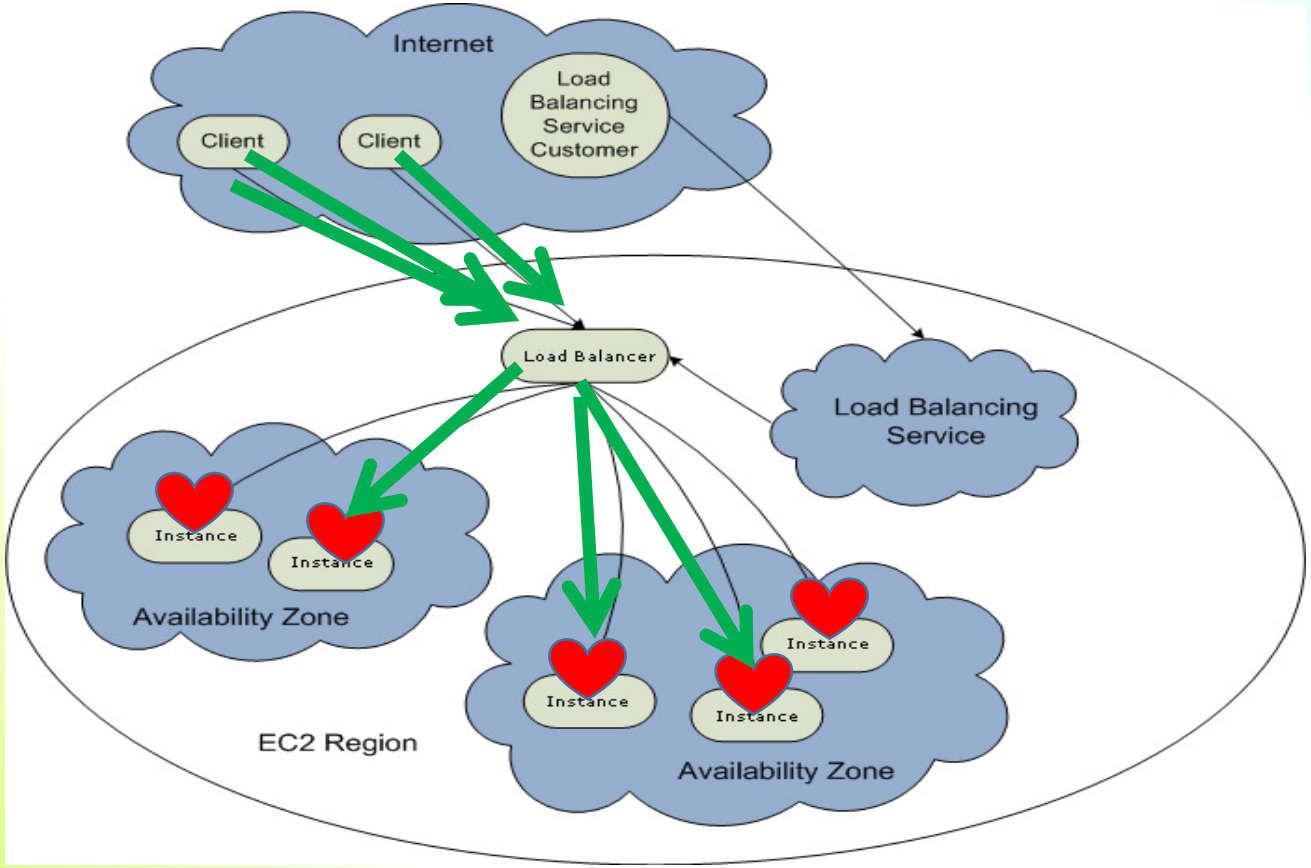
Introduction

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 + ε approximation)*
  4. *Random Algorithm*

Maximize Close User Amount

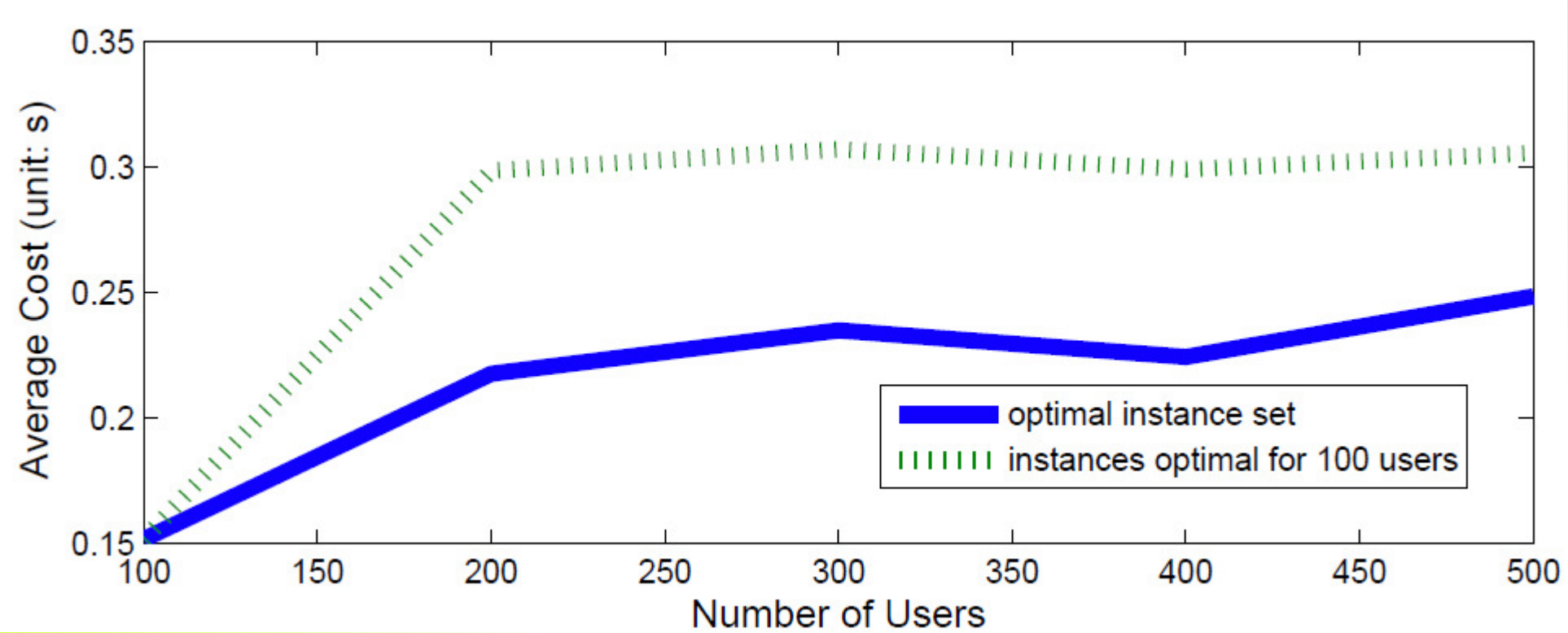
- *Max k-cover problem*
- NP-hard
- W[2]-hard with k as parameter
- W[2]-hard (general) and FPT (tree-like) with maximum subset size as parameter
- FPT if both maximum subset size and capacity as parameter

Maximize Close User Amount

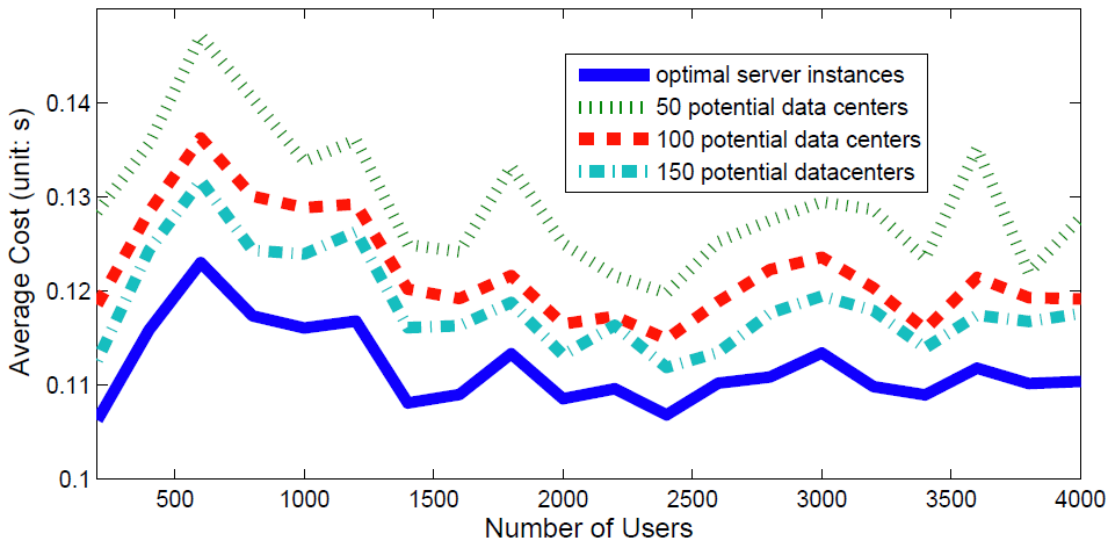
- Approximate Algorithms:
  1. *Greedy Algorithm (1-1/e approximation)*
  2. *Local Search Algorithm*

Single Service – Necessity of Redeployment

- Worst case without redeployment

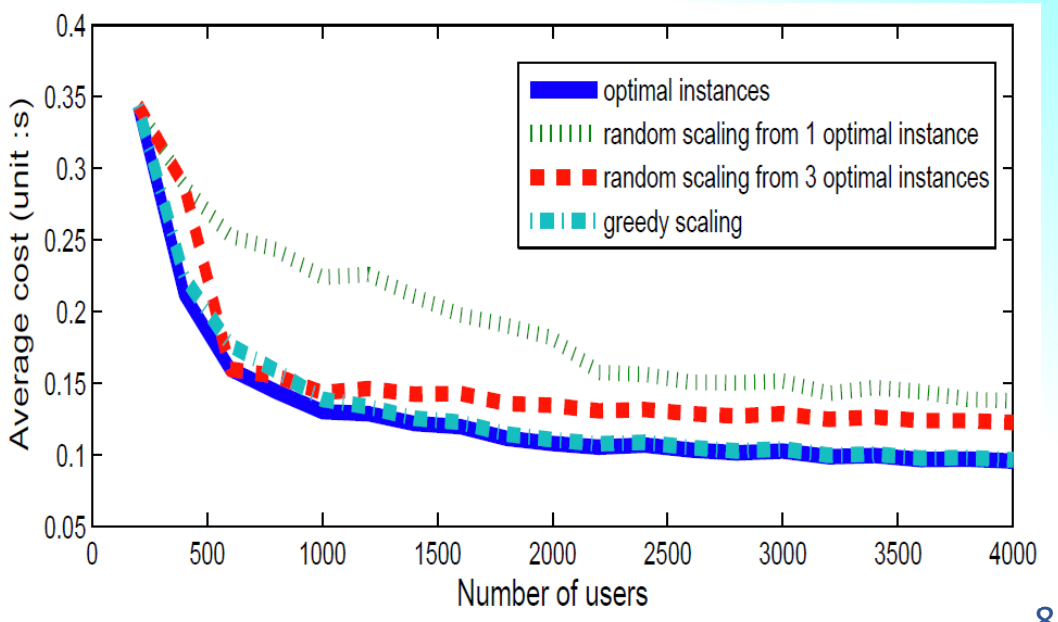


Single Service – Weakness of Auto Scaling

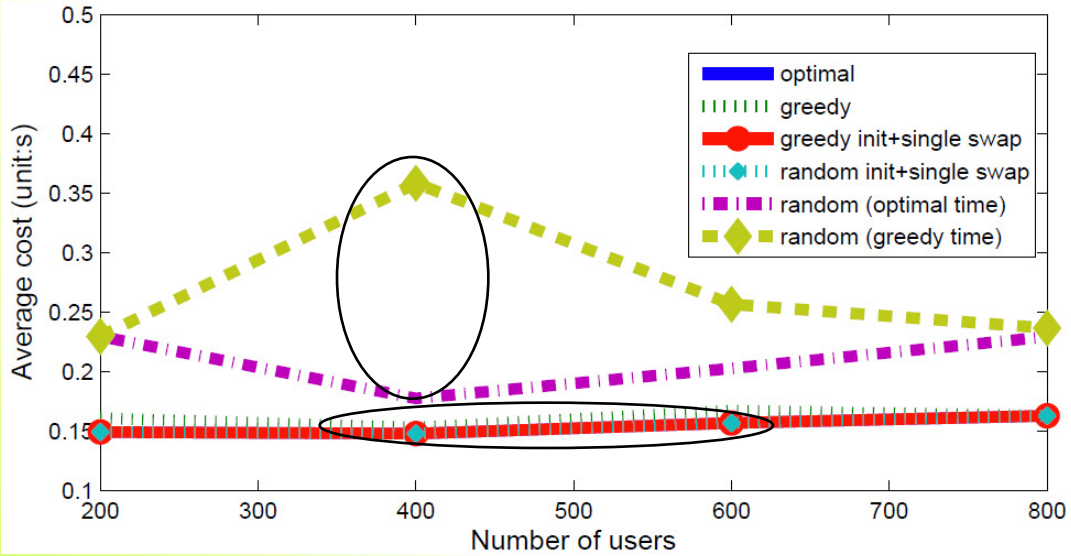


Deploy in limited data centers

Auto scaling algorithms

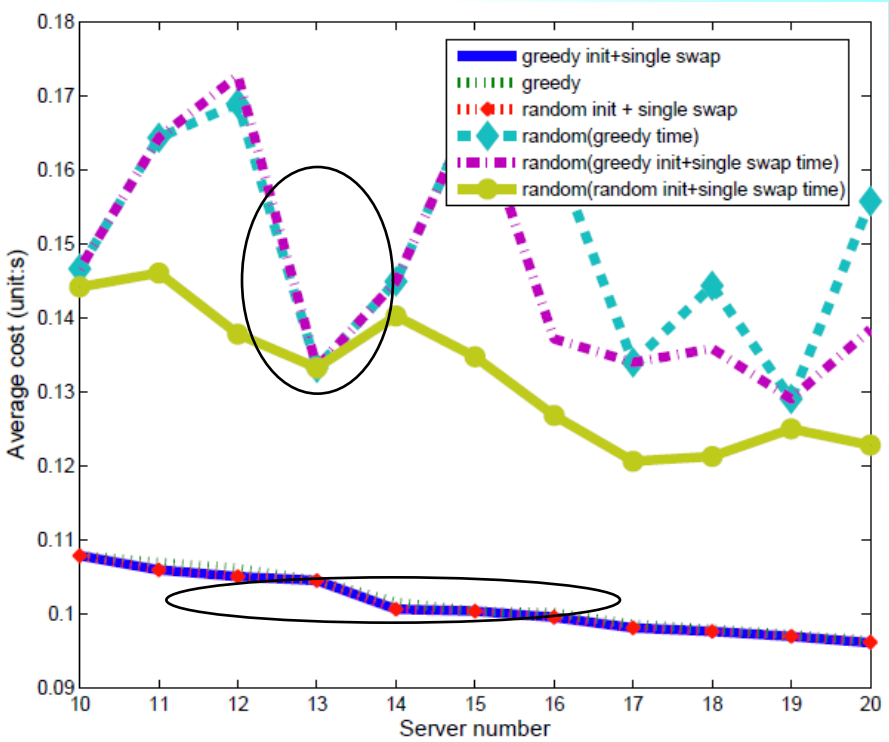


Single Service - Comparing Algorithms for k-Median



Selecting 3 data centers by redeployment algorithm

Selecting 10-20 data centers for 4000 users



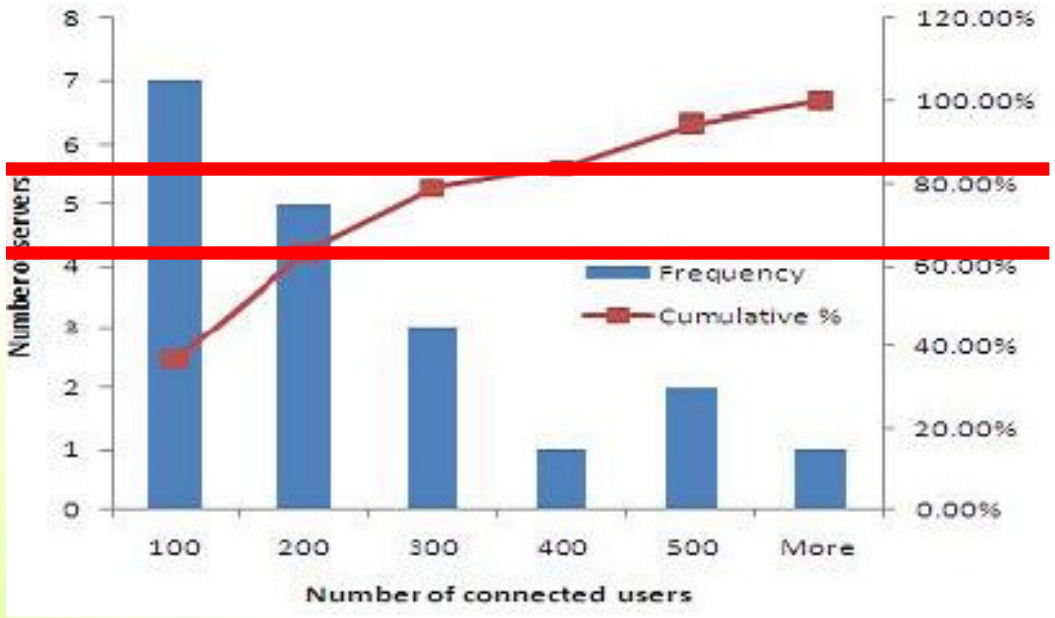
Single Service - Comparing Algorithms for k-Median

	Brute Force	Greedy	Greedy init + single swap	Random init + single swap
(2,200)	203	< 1	< 1	16
(2,400)	375	< 1	16	31
(2,600)	547	< 1	15	47
(2,800)	735	< 1	31	31
(3,600)	78969	< 1	31	63
(10,4000)	-	94	328	2641
(15,4000)	-	172	500	13109
(20,4000)	-	203	1906	25469

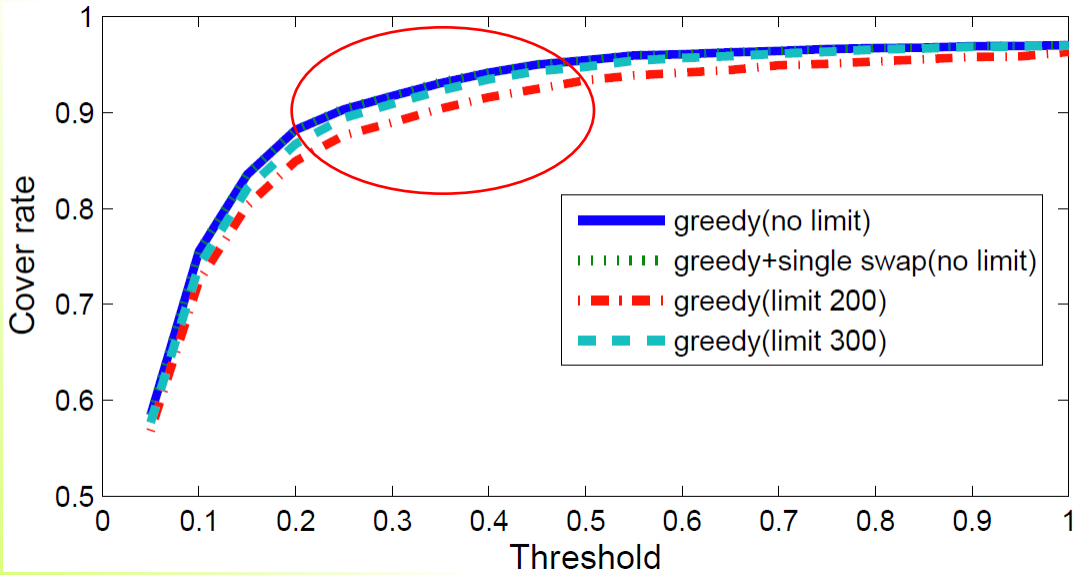
- Theoretical time complexity
  - Exhaustive search: $O(M^k \cdot N)$
  - Greedy: $O(k \cdot M \cdot N)$
  - Local Search: $O(k^t \cdot M^t \cdot N)$

Single Service - Redeployment Algorithms for Max k-Cover

- 20 instances are selected to provide service for 4000 users.
- Expect 200 per server.



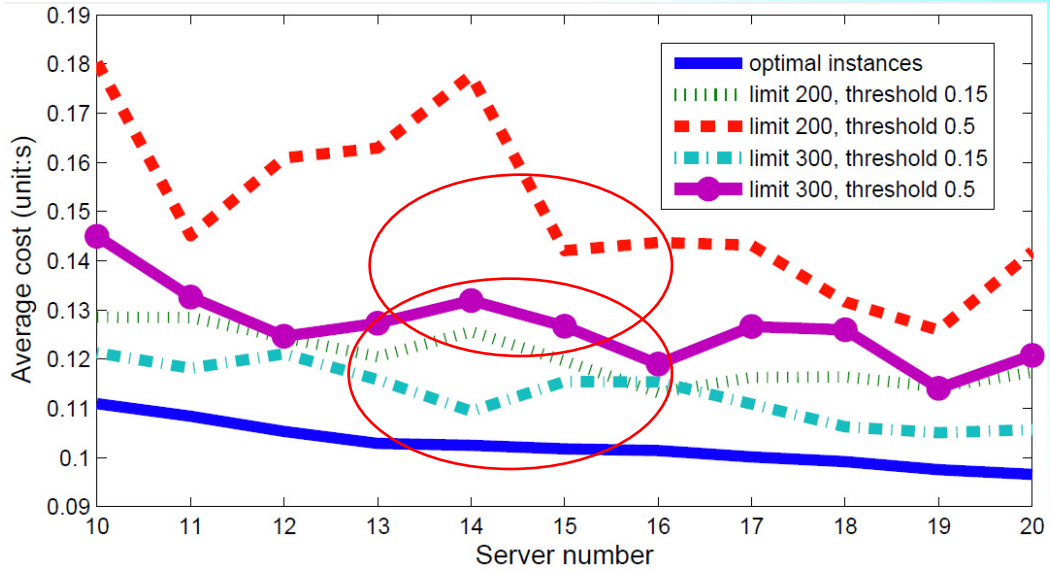
Single Service - Redeployment Algorithms for Max k-Cover



Max k-cover using greedy approach

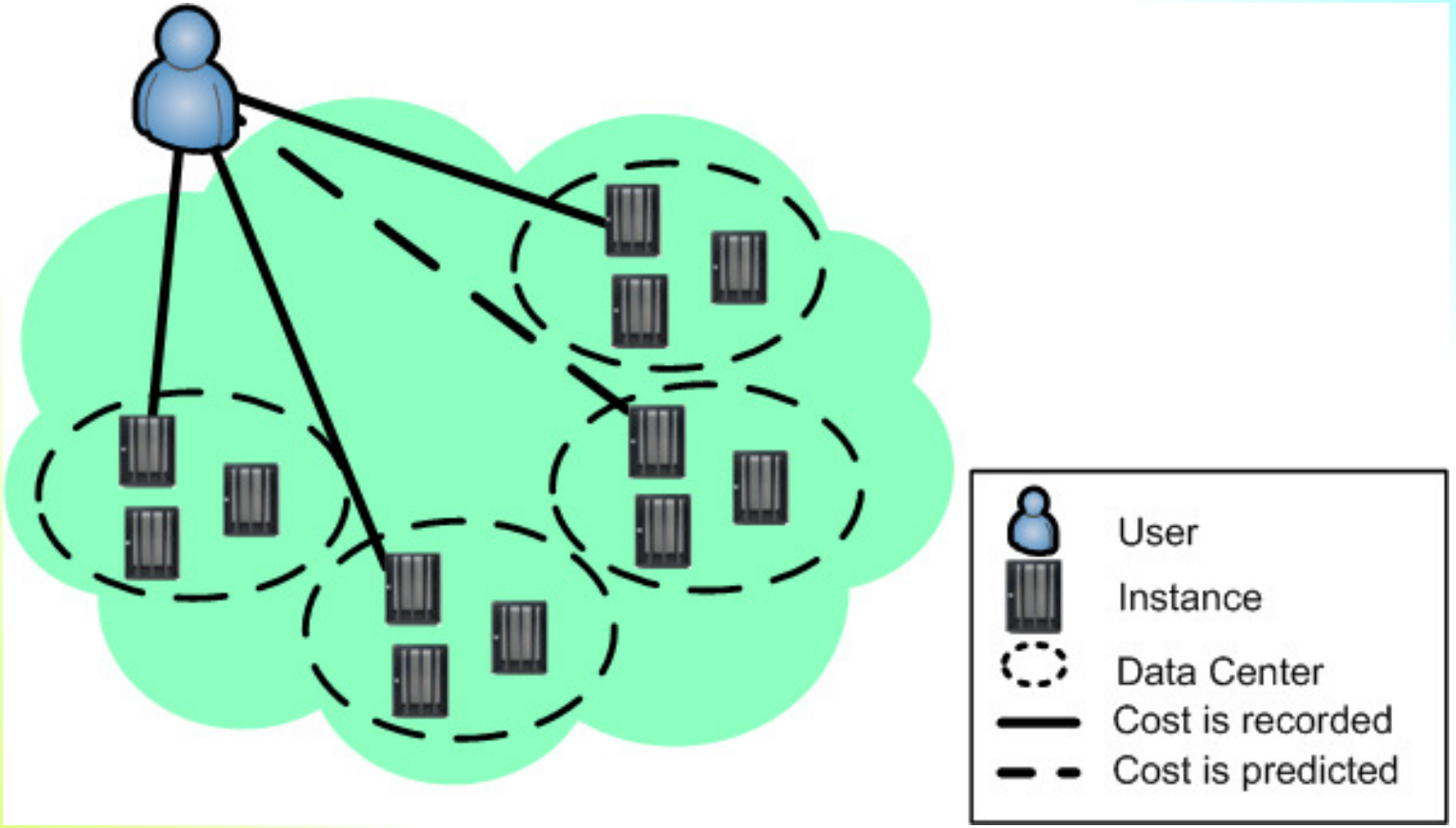
- **compare the average cost:** ***max k-cover v.s. k-median***

Average cost by max k-cover model

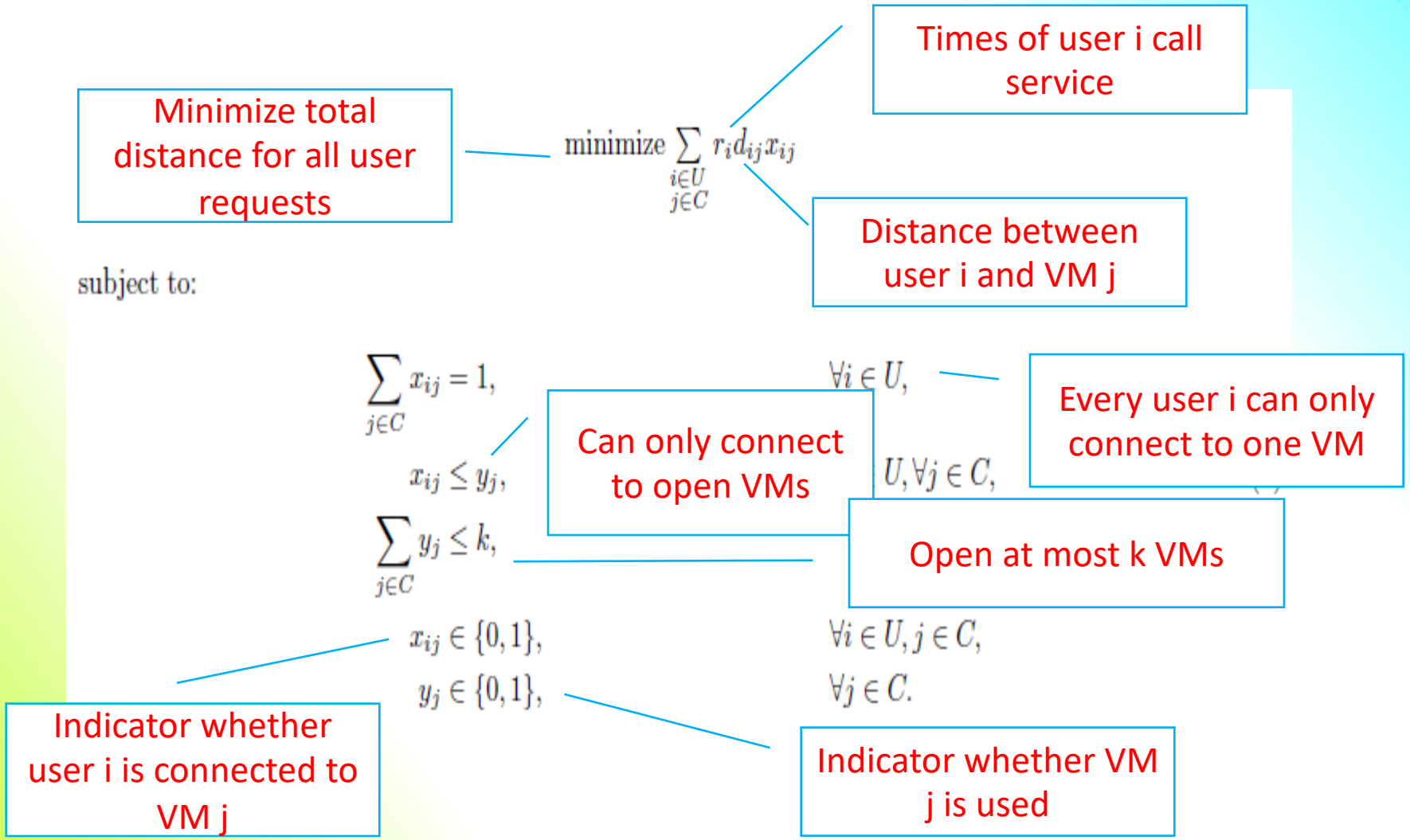


Multi-Service Deployment

Independent Deployment of Single Service



Single Service Deployment



$$\text{minimize} \sum_{\substack{i \in U \\ j \in C}} r_i d_{ij} x_{ij}$$

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

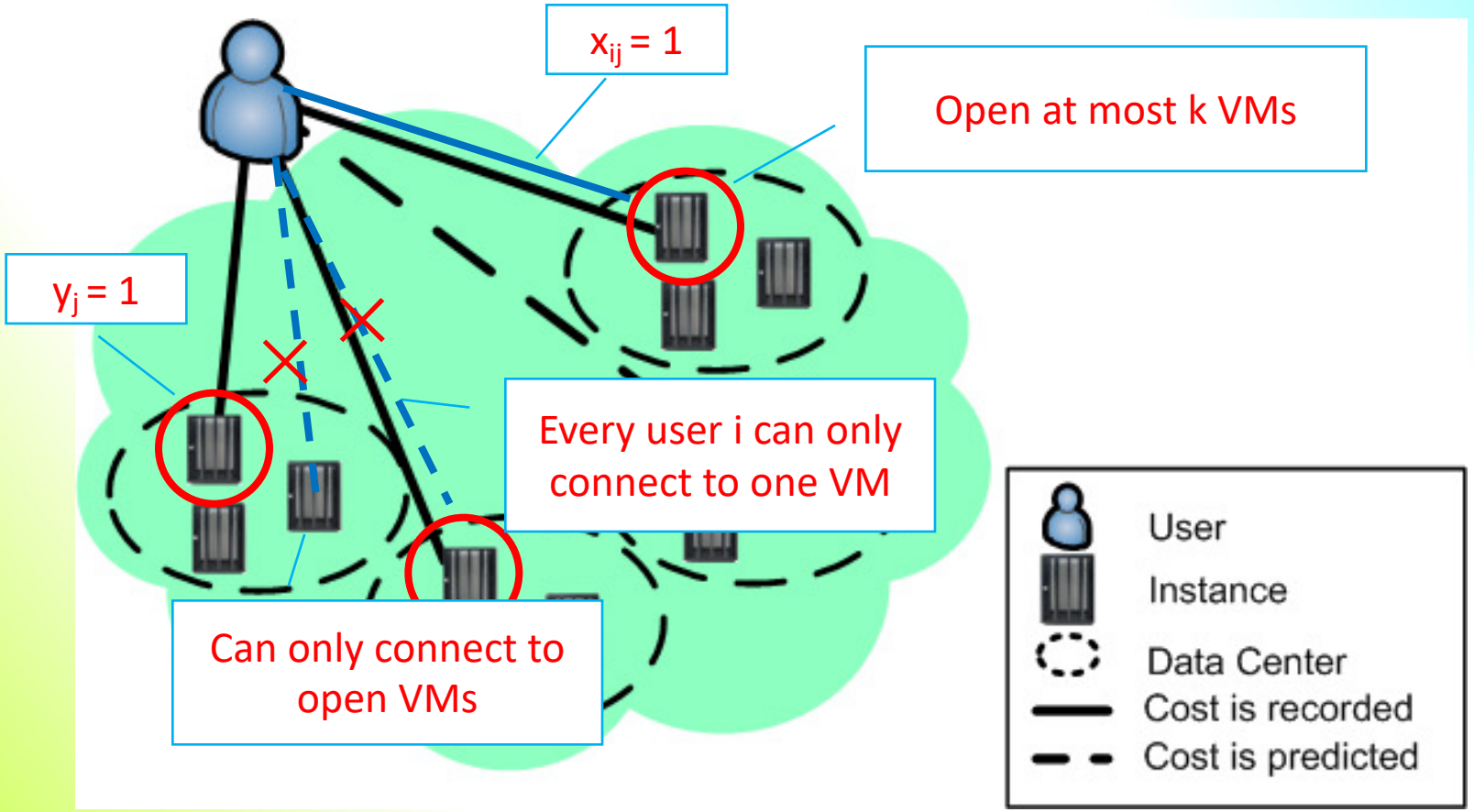
$$x_{ij} \leq y_j, \quad U, \forall j \in C,$$

$$\sum_{j \in C} y_j \leq k,$$

$$x_{ij} \in \{0,1\}, \quad \forall i \in U, j \in C,$$

$$y_j \in \{0,1\}, \quad \forall j \in C.$$

Independent Deployment of Single Service



Multi

$$\text{minimize} \sum_{\substack{i\in U\\ 1\le h\le m\\ i\in C_h}} r_{hi}d_{hij}x_{hij} + \sum_{\substack{i\in U\\ 1\le q,s\le m, q\ne s\\ p\in C_q, r\in C_s}} r_{iqs}d_{pqrs}y_{ipqrs}$$

subject to:

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

$$x_{hij} \le z_{hj}, \tag{2}$$

$$\sum_{j\in C_k} z_{hj} \le k_h, \tag{3}$$

$$\sum_{\substack{1\le s\le m\\ s\ne h\\ r\in C_s}} y_{ijhrs} \le x_{hij}, \tag{4}$$

$$y_{ipqrs} \le z_{rs}, \quad \forall i \in U, \ 1 \le q, s \le m, q \ne s, \ \forall p \in C_q, \forall r \in C_s, \tag{5}$$

$$\sum_{\substack{p\in C_q\\ r\in C_s}} y_{ipqrs} = sign(r_{iqs}), \tag{6}$$

$$x_{hij} \in \{0,1\},$$

$$y_{ipqrs} \in \{0,1\},$$

$$z_{hj} \in \{0,1\},$$

Times of user i call service h

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

connect at most one center

limitation # of instances every service

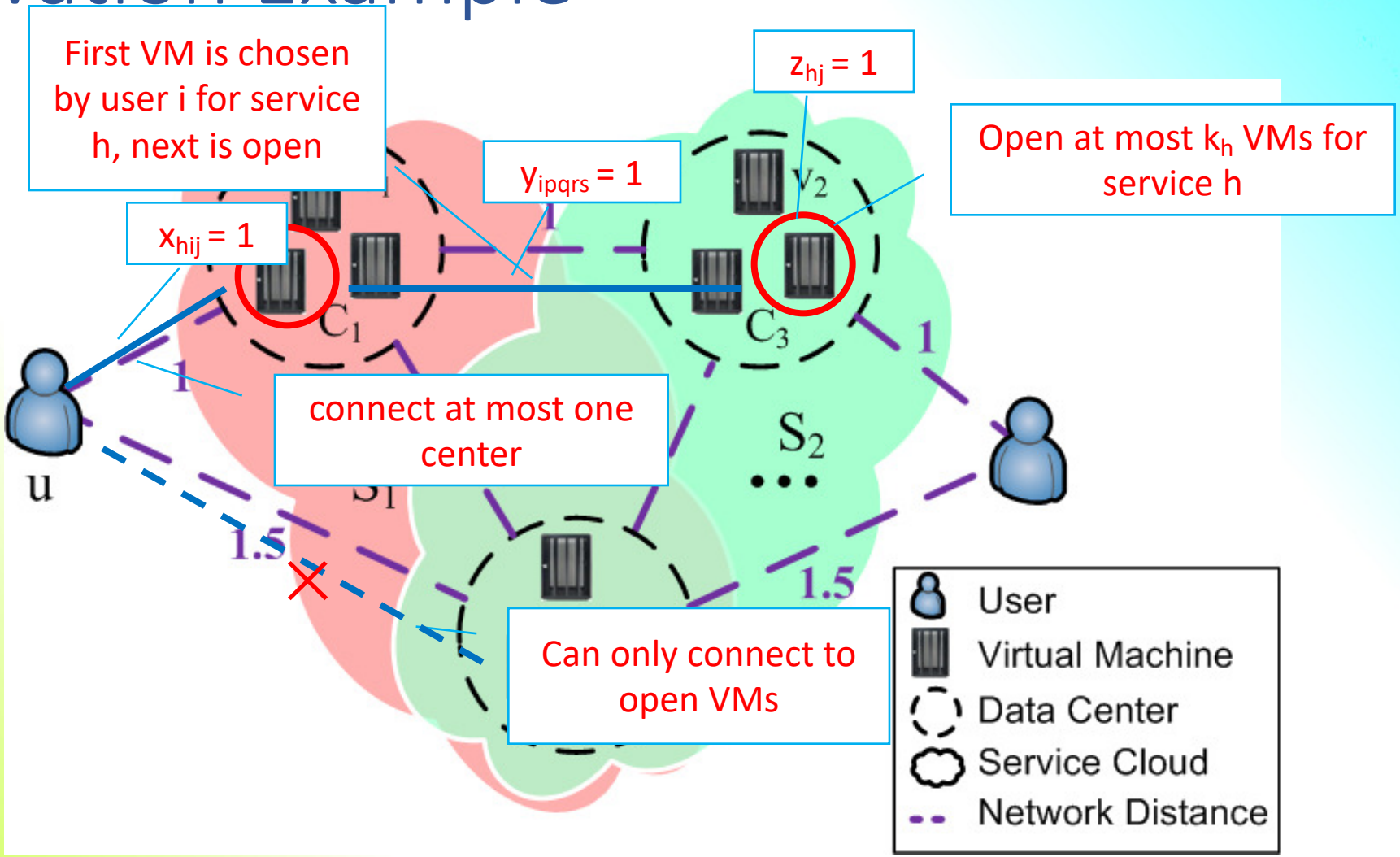
both centers should be opened

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



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 a
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 these from users

Record the best till now

Disturb and do local search

Dataset Statistics

Statistics	P2W Matrix	P2P Matrix
minimum	0.01	0.08
$25th$ percentile	53.69	58.58
median	118.14	133.76
$75th$ percentile	176.00	188.53
maximum	1604.02	5704.04
average	129.41	138.38

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 -> service s_{i1} -> service s_{i2} -> … -> service s_{im}

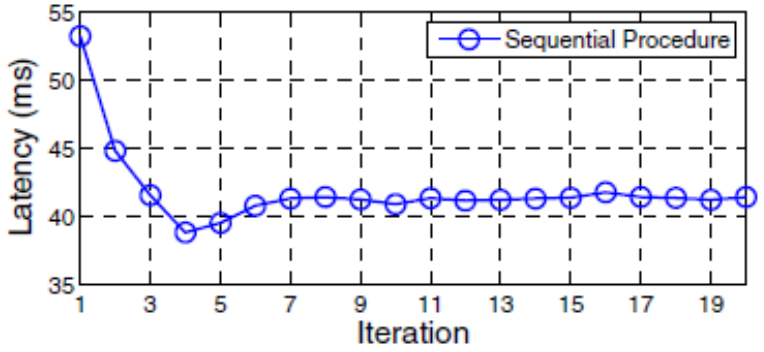
Notation	Descriptions	Default
n_h	Number of log entries of service h	1880
ρ_h	Ratio of users use service h to total users	0.2
l	Number of services involved in one service request	5

Default Experiment Setting

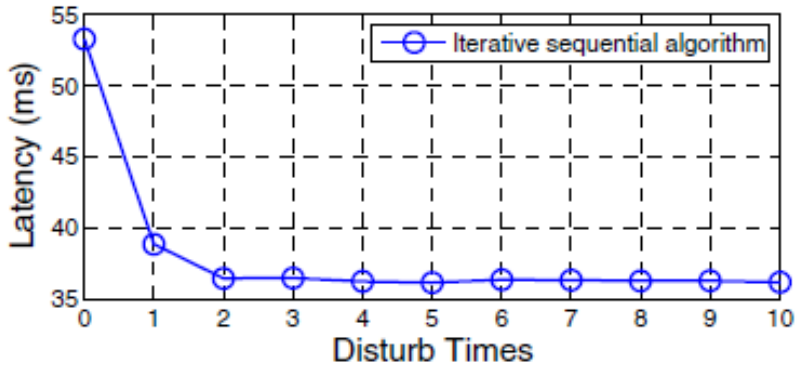
- 1881 users
- 10 services
- Deploy10 service VMs among a candidate set in 100 data centers
- A user of service s *would have 5 request logs*
- *One request* of a service would involve on average 5 requests of other services

Experiment (Algorithm Specifics)

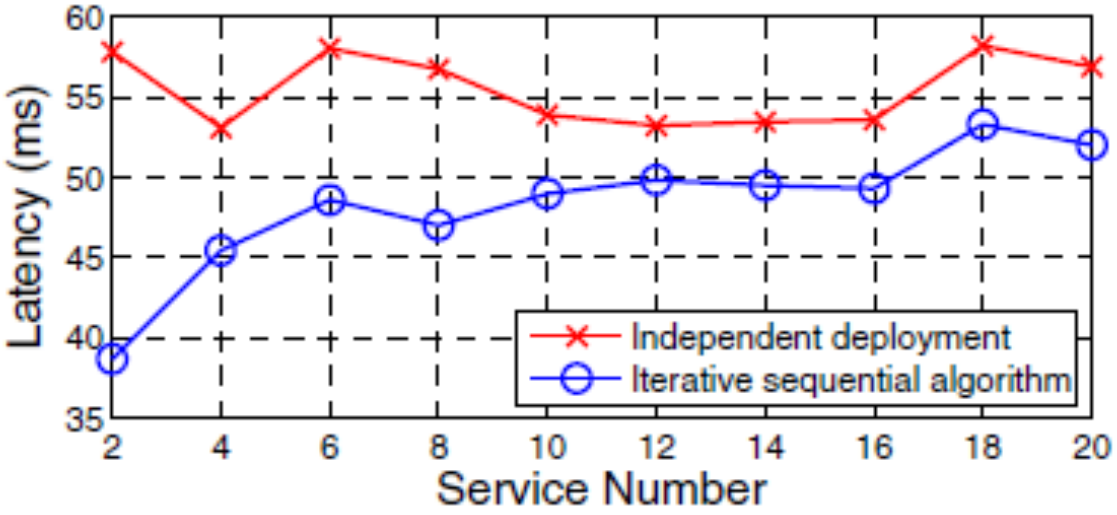
- *Convergence of Iterative Sequential Procedure*



- *Number of Disturbs*

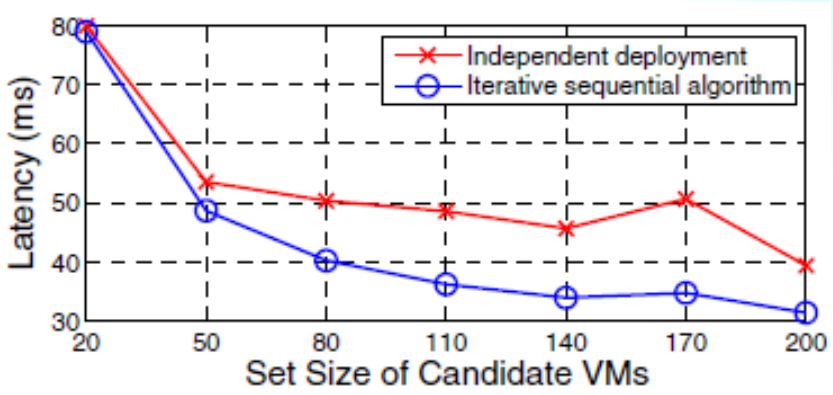


Experiment (*Number of Services)*

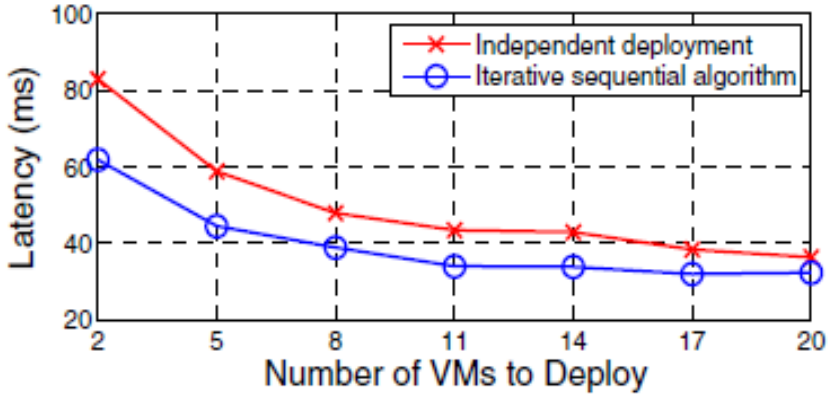


Experiment (*Number of Service VMs)*

- *Size of Candidate Set of Service VMs*

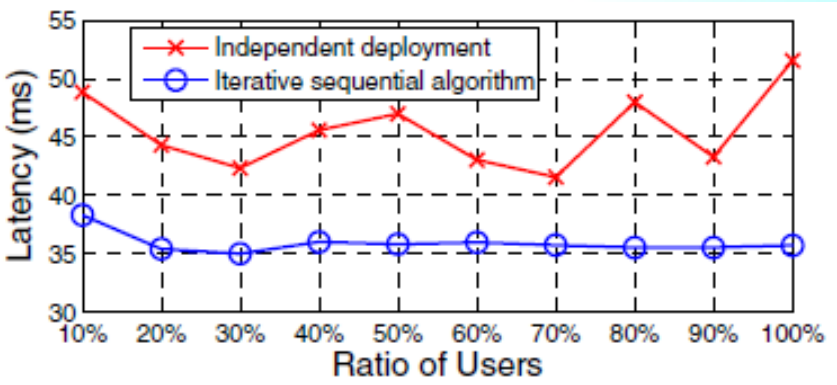


- *Number of Service VMs to Deploy*

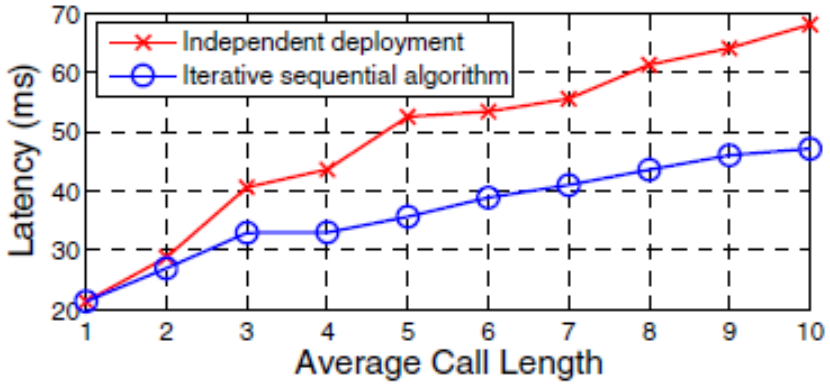


Experiment (*Services Logs)*

- *Number of Service Users*

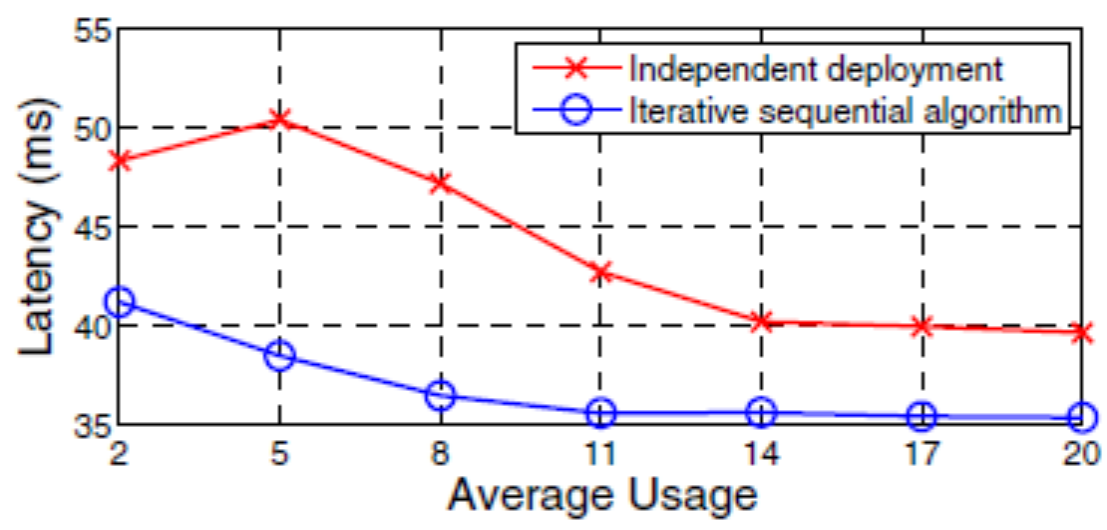


- *Average Call Length of Service*



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

Publication List

1. “Pretect: Detecting Poor-Responsive UI in Android Applications”, **Yu Kang**, Yangfan Zhou, Min Gao, Yixia Sun, Michael R. Lyu, submitted to ISSRE 2016
2. “DiagDroid: Android Performance Diagnosis via Anatomizing Asynchronous Executions,” **Yu Kang**, Yangfan Zhou, Hui Xu and Michael R. Lyu, accepted by *FSE 2016*
3. “SpyAware: Investigating the Privacy Leakage Signatures in App Execution Traces,” Hui Xu, Yangfan Zhou, Cuiyun Gao, **Yu Kang**, and Michael R. Lyu, in ISSRE 2015
4. “Dependability Issues of Android Games: A First Look via Software Analysis,” Jiaojiao Fu, Yangfan Zhou, and **Yu Kang**, in IVCE 2015
5. “A Latency-aware Co-deployment Mechanism for Cloud-based Services,” **Yu Kang**, Zibin Zheng, and M.R. Lyu, in Cloud 2012
6. “WSP: A Network Coordinate based Web Service Positioning Framework for Response Time Prediction,” Jieming Zhu, **Yu Kang**, Zibin Zheng, and M.R. Lyu, in ICWS 2012
7. “A Clustering-Based QoS Prediction Approach for Web Service Recommendation,” Jieming Zhu, **Yu Kang**, Zibin Zheng, and M.R. Lyu, in IVCE 2012
8. “A User Experience-based Cloud Service Redeployment Mechanism,” **Yu Kang**, Yangfan Zhou, Zibin Zheng, and M.R. Lyu, in Cloud 2011