Hindawi Publishing Corporation
Mobile Information Systems
Volume 2016, Article ID 7462821, 7 pages
http://dx.doi.org/10.1155/2016/7462821

Hindawi

Research Article

Energy-Reduction Offloading Technique for Streaming Media Servers

Yeongpil Cho,[1] Oparin Mikhail,[1] Yunheung Paek,[1] and Kwangman Ko[2]

[1]*Department of Electrical and Computing Engineering, Seoul National University, Seoul, Republic of Korea*
[2]*School of Computer and Information Engineering, Sangji University, Seoul, Republic of Korea*

Correspondence should be addressed to Kwangman Ko; kkman@sangji.ac.kr

Received 16 August 2015; Revised 25 November 2015; Accepted 26 November 2015

Academic Editor: Javid Taheri

Recent growth in popularity of mobile video services raises a demand for one of the most popular and convenient methods of delivering multimedia data, video streaming. However, heterogeneity of currently existing mobile devices involves an issue of separate video transcoding for each type of mobile devices such as smartphones, tablet PCs, and smart TVs. As a result additional burden comes to media servers, which pretranscode multimedia data for number of clients. Regarding even higher increase of video data in the Internet in the future, the problem of media servers overload is impending. To struggle against the problem an offloading method is introduced in this paper. By the use of SorTube offloading framework video transcoding process is shifted from the centralized media server to the local offloading server. Thus, clients can receive personally customized video stream; meanwhile the overload of centralized servers is reduced.

1. Introduction

Smartphones popularity in past few years has grown increasingly. According to Strategy Analytics, a market research company specializing in digital products and electronics, end users have bought around 1 billion smartphones in the third quarter of 2012 compared to 700 million units which had been sold in 2011. The popularity can be explained by increasing abilities of mobile devices, where these days smartphones are mostly used for games, social networks, and videos. One of the most promising and convenient ways for users to deliver multimedia content in the Internet is multimedia streaming. Already today multimedia streaming takes up about 80% of video data traffic and engages for even bigger share in future. Heterogeneity of modern mobile devices, that is the difference in screen size and computational power of each concrete model of mobile devices along with their energy limitations, necessitates customized video transcoding for all end user devices with the unique parameters [1, 2]. Also, different devices can have different Internet bandwidth, which imposes additional requirements upon the quality of streaming video and therefore customized transcoding [3].

The abovementioned can lead to several issues in work of media servers, namely, network overload, storage space shortage, and computational overload of media servers. The first problem, caused by increasing amount of clients trying to access a media server, can be solved by organizing the network in decentralized or distributed manner. Various approaches to solve this issue have been suggested, among which most notable are the decentralization concerning subscribers [4] and an approach with the use of proxy servers [5]. The objective of this paper is to address the problem of computational overload of media servers. So far, the existing researches concern moving of the transcoding process, the most computation-intensive part of video streaming, to other accessible resources. In [2] transcoding is done by adaptation engine which is a solution for multimedia data customization problem rather than the media server overload problem. In [6, 7] adaptive transcoding performed at proxy servers or video adaptation nodes located on the edges of fixed networks, while in [8, 9] transcoding is fairly spread among existing neighbor servers. Although those approaches reduce the overload of media servers, they still might encounter difficulties in case huge number of video files need to be

transcoded. Another drawback of these studies is that if a user has several heterogeneous devices and needs the video for each of them, the only way to get customized video files is to access the media server several times, which again emphasizes the media server overload problem. Thus, the more efficient way to address the problem would be to perform transcoding not at server side but at client side. However, in view of the client side being a portable device with limited power and energy resources this solution does not seem viable. Therefore, to reduce the overload of media servers while keeping the transcoding process at client side a novel mobile offloading approach inspired by cloud computing is introduced in this paper.

Mobile offloading [10, 11] is a powerful technique helping to move most power-consuming computations from resource-constrained portable devices to more powerful ones. The main difference between mobile offloading and client-server approaches is that in case of the second one a client always migrates some part of work to the server [12], while in case of mobile offloading it is decided dynamically which parts of code should be run at the server side. One of notable studies on offloading is [13], where a solution for optimal profile-based partitioning of source code for sensor network application is demonstrated. Another revealing work is [14]. It demonstrates partitioning of graphical applications with the use of adaptive offloading decisions.

To implement the above-described idea, a SorMob offloading framework [15] along with SorTube multimedia streaming cross-platform application has been used. SorMob framework does mobile offloading with the use of Aspect-Oriented Programming (AOP) paradigm for separation, weaving, and reuse of the code marked by cross-cutting concerns [16]. It makes the offloading implementation flexible and fully intuitive for a programmer. SorTube's job is to adapt a video file for the end device and then stream it to the client and view it at the client side. All of these should be done simultaneously so that the user can view the video with the least delay.

The rest of the paper is organized as follows. Sections 2 and 3 demonstrate the architectures of SorMob and SorTube. In Section 4 the results of experiments are shown. And Section 5 concludes and traces a course of future studies.

2. SorMob

SorMob is an AOP-implemented source-level offloading framework, which moves part of computation to the cloud, based on source code profiling [17]. It calls the methods placed at the offloading server through the Remote Procedure Calls (RPC) so that a programmer does not need to explicitly code the details for this interaction. In what follows, preliminary introduction of AOP basics along with the architecture and implementation of SorMob is introduced.

2.1. Aspect-Oriented Programming. One of the first and major works on AOP programming technique was done in [16], where aspects and a new way of clear expressing programming involving such aspects were introduced. Basically, AOP is a code injection; only it deals with concerns that cut across

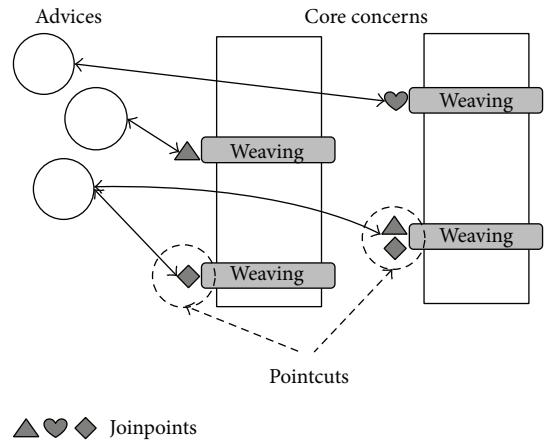

Figure 1: AOP concept.

the whole application. Base units in AOP concept are *advices*. Advices are the parts of the code which are able to be added to specified points in the program when it executes. The specified points in the program are called *joinpoints* (JPs) and normally are located before or after the methods in the code. Matching of JPs to the advices is determined by *pointcuts* which itself appears as a set of JPs. The process of adding advices to the source code is called *weaving* or *cross-cutting* and it can be done during compilation, loading, or execution time. Key points of AOP technique are represented in Figure 1, where various advices are weaved into the source code at the moments specified by *joinpoints* within pointcuts.

Because of its clear concept, the use of AOP in SorMob makes the implementation of the offloading framework simpler and more flexible. It also makes SorMob fully independent of applications specificity as long as these applications are written in Java. Additionally, before, we could choose one of two ways to implement offloading for Android applications: either to modify Dalvik Virtual Machine (DVM) and make the offloading process automated or to manually modify each application adding the code for offloading to it. Both ways are painful compared to AOP, which also provides automated offloading without any need to modify target applications. With AOP adding offloading functionality to an Android application is as easy as adding SorMob library to this application.

Algorithm 1 shows an example of marking the parts of the code for migration. A *doGame* method which is needed to be migrated is marked with *@Migratable* annotation in front of it. Therefore, when the annotated method is called, migration starts executing automatically to offload the method. Also, additional serialization mechanisms are needed in case of Java automatic serialization.

2.2. SorMob Architecture. An architecture and workflow of SorMob are shown in Figure 2. It consists of a migrator, profiler, solver, classloader, and offloader for server and client sides. Migrator serves to extract the annotated methods from the application source code. Profiler measures the execution time and appearance frequency of each method in the source

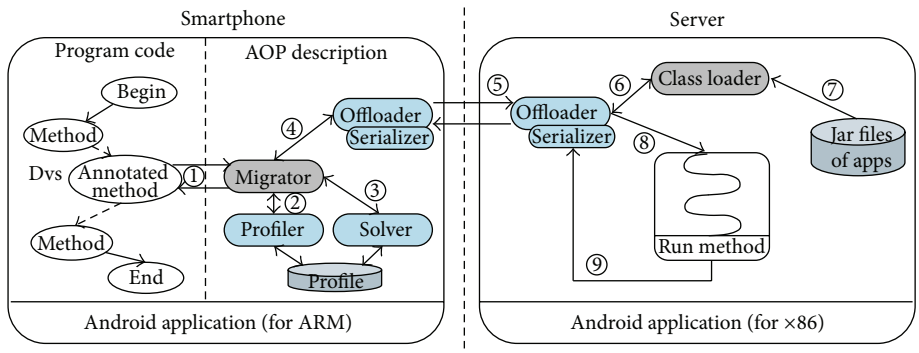

Figure 2: SorMob architecture and workflow.

```
public class MainActivity {
   GAME g = new GAME( );
   public void onClick(View v) {
      g.doGame("attack");
      ...
   }
   ...
}
public class GAME implements Serializable {
   Object [ ] enemies;
   @Migratable
   public Object doGame(String cmd) {
      ...
   }
   ...
}
```

Algorithm 1: Aspect-Oriented Programming methods annotation.

code, size of the state needed to be transmitted to offload the method, round-trip delay time (RTT), and network bandwidth. Solver makes a decision on the migration of each annotated method based on the information received from the profiler. Selected methods are sent to the server by client-side offloader, which in effect performs serialization of the data for transmission. The states needed to be sent to the server consist of classes, objects, and arguments of target methods as well as names of target objects. To serialize the data SorMob uses Kryo open-source serialization library, which helps to avoid the abovementioned inconvenience of Java built-in serialization.

When the offloader at server side receives the offloaded states, it keeps waiting until the classloader dynamically loads all the classes required for target methods. Those classes have to be installed at the server side in advance. After execution of the methods at server side, the offloader collects only the states which have been changed, thereby reducing the amount of redundant data needed to be sent back to the client. At client side, sent and received byte arrays are compared using difference algorithm, and the result is passed to the application, which then can continue executing.

3. SorTube

3.1. SorTube Architecture. SorTube is a cross-platform streaming media player application with video customization for heterogeneous devices. The workflow of SorTube is shown in Figure 3. The application consists of a server and client parts, which are identical, apart from the fact that they are built for different platforms. Server-side SorTube is built for ×86 architecture, while client-side SorTube is built for ARM architecture. Inner structure of SorTube application contains three main elements: transcoding module, streaming module, and video player. Because all three modules need to run at the same time, each of them is organized as a separate thread.

As opposed to general case when a video file is simply streamed from media server to client, in case of SorTube implementation, the offloading server is also involved. At first, the video is streamed from remote media server to the offloading server. There, video goes through transcoding module, which adjusts its parameters such as video format, bit rate, resolution, and pixel format, to make the video finely fit the end user device. After getting customized video passes to streaming module, which sends it to the client. When client-side SorTube receives the video stream, video player starts viewing it right away. All the above-described steps go on simultaneously allowing the user to view the video without waiting for completion of video data transfer.

Implementation of SorTube is based on open-source projects. Transcoding module is based on FFmpeg, a cross-platform multimedia converting library containing the majority of existing codecs. Transcoder also contains H.264 encoder to be able to produce end video supported by Android operation system. The encoder is implemented based on ×264 library. Output of the transcoder is a flow of video chunks, where the length of chunks determines the granularity of video streaming. The less the length, the less the delay between the request and viewing of the video. Streaming module is implemented with the use of Netty, event-driven network application framework. Chunks transfer between offloading server and mobile device is organized using client-server model and HTTP protocol. When chunks reach client side, they are put together and stored as one video file, which imitates progressive download

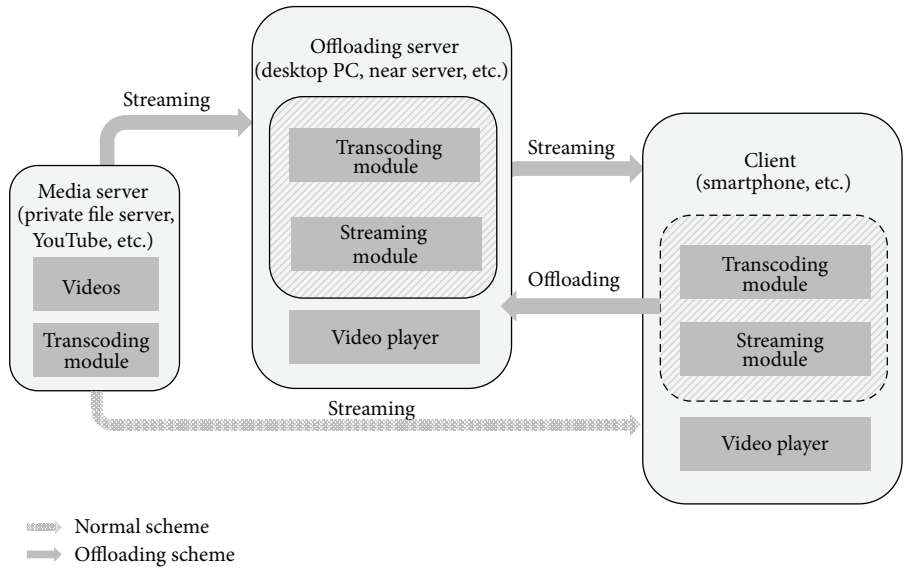

Figure 3: SorTube workflow.

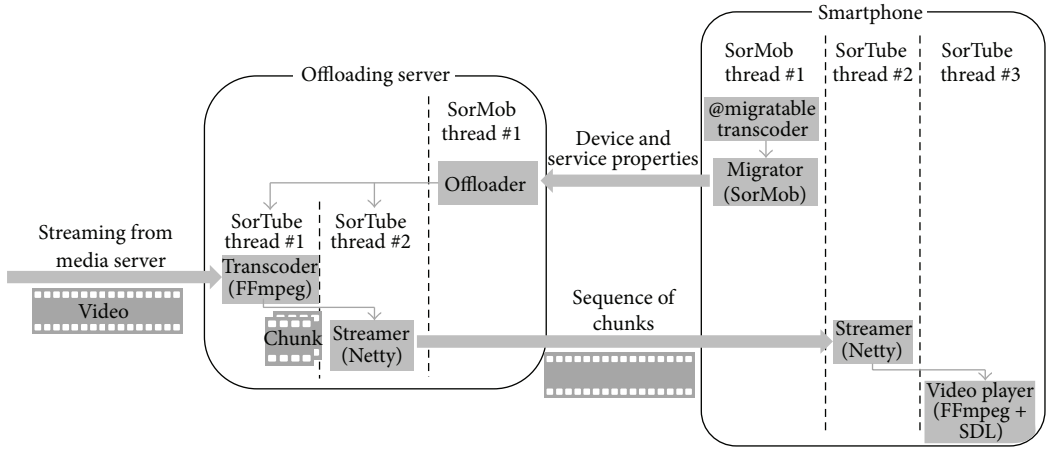

Figure 4: Integration of SorTube and SorMob.

method for media data transfer. Video player is implemented with SDL, a cross-platform multimedia library for MPEG playback software, added to FFmpeg. The player continuously accesses the file streamed by offloading server and as soon as new chunks arrive, it views them.

3.2. SorTube-SorMob Integration. Application-level integration of SorTube and SorMob is demonstrated in Figure 4. At client side SorMob runs as a separate thread, which can be accessed by SorTube in case the annotation for migration is met in SorTube's source code. SorTube passes the address of possible offloading server to SorMob framework and if the connection between the mobile device and the server can be established, migration of transcoding occurs. Along with the object states needed for running transcoding module at the offloading server, required parameters of the end video file are sent at migration time.

At the offloading server side, SorMob is organized as the main thread which initiates SorTube's modules in case migration request from SorTube occurs. If there is no migration request from SorTube, SorMob processes migration requests from other applications or stays in idle state.

4. Experiments

4.1. Methodology and Configurations. To evaluate the efficacy of SorTube-SorMob solution several questions need to be answered. First, the relation between the growth of number of clients accessing a media server and the overload of the media server needs to be estimated. Also, how the overload of the media server affects the quality of service for the clients needs to be evaluated. To support the idea of using offloading server for transcoding, inviability of performing it at the mobile device side should be demonstrated. Finally, an improvement

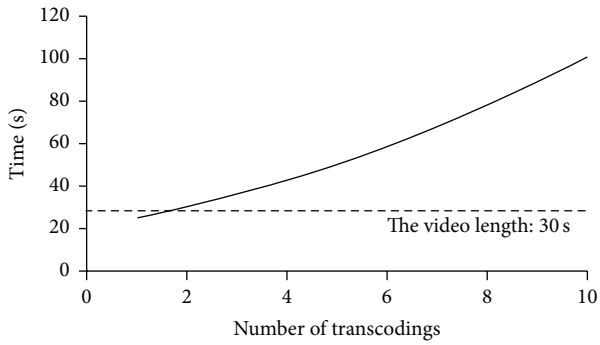

Figure 5: Transcoding execution time for changing number of clients.

in power consumption, achieved due to video customization, needs evaluation.

To conduct the experiments, a smartphone with 1.2 Ghz CPU and 1 GB RAM running Android 4.0.3 version was used as the end user device. Two desktop PCs, with 3.1 GHz quad-core CPU and 8 GB RAM each, represented the media and offloading servers. Both desktops used Linux OS, while the offloading server also ran Android environment in a virtual machine (VM).

4.2. Measurement of Media Server Overload. For quantitative assessment of possible media server overload, several VMs ran at the desktop representing offloading server. All VMs performed transcoding of the same video file simultaneously. Depending on the number of active VMs, the change in time required to finish transcoding was measured. In the course of the experiment, a 30-second-long video precoded with H.264/MPEG-4 AVC video codec at 1.8 Mbps bitrate was transcoded. A resolution of the video was changed from 1280 × 720 to 480 × 272 pixels. Figure 5 depicts the results of the experiment for a range of clients varying from 2 to 10.

It can be seen from the plot that when the number of devices becomes more than two, the time needed to finish transcoding is getting longer than the time length of the video file itself. Although transcoding time depends on the performance of the media server, even for most powerful ones, there exists the number of clients, so that the time required for transcoding will be greater than the length of the video. In this case, noticeable delays during video viewing will occur affecting the quality of media streaming service.

4.3. Transcoding by Mobile Device. A straightforward solution for relieving media servers can be to localize transcoding of a video to the client side, that is, to move it from a media server to the mobile device requesting the video. As a result, media servers do not need to do transcoding anymore and can send the requested video to the client right away. However, computational power of the mobile device is much lower compared to the one of the media server, what makes transcoding last significantly longer (Table 1). Besides, transcoding process consumes a great amount of energy. For example, to transcode the original video file to the resolution of 480 × 272 pixels took about 20 mAh. Extending the results for 1-hour-long video, apparently the capacity of the mobile device battery will not be enough to finish such transcoding.

Table 1: Mobile device transcoding.

Video parameters	Time (s)	Avg. power consumption (mW)	Energy consumption (mAh)
480 × 272.2 Mbps	123.84	2311.81	21.49
960 × 545.2 Mbps	243.48	2206.07	40.33
240 × 186.2 Mbps	53.16	2586.28	10.32
480 × 272.4 Mbps	149.07	2119.70	23.72
480 × 272.1 Mbps	105.79	2353.33	18.69

4.4. Transcoding by Offloading Server. An alternative to the mobile device transcoding, which also keeps the process localized at client side, is to perform it on the offloading server. In this case, total energy consumed by mobile device can be separated into three parts: the energy spent to request a video file, the energy spent to migrate transcoding and to receive streamed video file, and the video viewing energy. Experimental results, showing energy consumption for each part as well as total energy consumption of SorTube application, are given in Table 2. This table shows that total energy consumption in case of offloading is substantially less than the one spent to perform only transcoding on the mobile device. Also, the energy consumed during streaming or offloading is relatively small compared to the energy spent for playing back. Since playback energy consumption decreases considerably over worsening of video resolution, it can be inferred that the use of the offloading server as a transcoder is more energy efficient than viewing video without customizing its parameters.

5. Conclusions and Discussion

In this paper a solution, called SorTube-SorMob, for reducing the overload of media servers was presented. It was shown that not only is the solution more effective compared to the previous studies in the area, but also it gives an improvement in a mobile device energy consumption while viewing videos on the Internet. To demonstrate the achievements of this work, three experiments were conducted. They corroborated that the overload of servers badly affects the quality of Internet media services and causes significant delays for clients to access the data; the most time and resource consuming process of transcoding cannot be migrated to mobile devices because of their limited computational and energy resources; the efficacy of using offloading server compared to simple viewing of video files with improper resolution was demonstrated.

SorTube-SorMob also has its drawbacks. One of them is that offloading framework uses annotated source code, because of which it cannot be applied to any application downloaded from the market. Also, for now, any annotated code is migrated to the offloading server if possible. As a future work dynamic decision-making mechanism on

Table 2: Offloading server transcoding.

(a) Summary values of energy, average power, and time consumption for offloading, streaming, and viewing

Video parameters	Time (s)	Avg. power consumption (mW)	Energy consumption (mAh)
480 × 272.2 Mbps	35.03	2005.88	5.28
960 × 545.2 Mbps	75.54	2171.14	12.31
240 × 186.2 Mbps	33.51	1965.72	4.95
480 × 272.4 Mbps	35.83	2114.71	5.69
480 × 272.1 Mbps	33.16	1988.58	4.95

(b) Values of energy, average power, and time consumption for video streaming only

Video parameters	Time (s)	Avg. power consumption (mW)	Energy consumption (mAh)
480 × 272.2 Mbps	17.04	1076.11	1.47
960 × 545.2 Mbps	34.07	1148.08	2.54
240 × 186.2 Mbps	16.67	991.28	1.50
480 × 272.4 Mbps	23.31	1201.35	2.22
480 × 272.1 Mbps	16.49	1268.12	1.39

(c) Values of energy, average power, and time consumption for video viewing only

Video parameters	Time (s)	Avg. power consumption (mW)	Energy consumption (mAh)
480 × 272.2 Mbps	30.00	1523.14	3.75
960 × 545.2 Mbps	70.94	2036.18	9.72
240 × 186.2 Mbps	30.00	1429.71	3.39
480 × 272.4 Mbps	30.00	1511.93	3.41
480 × 272.1 Mbps	30.00	1529.93	3.87

(d) Values of energy, average power, and time consumption for state offloading. Keeps the same over videos with different parameters

Time (s)	Avg. power consumption (mW)	Energy consumption (mAh)
3.87	1047.30	0.06

migration, considering the cost of offloading and possible outcomes, will be implemented. Additionally, if a user has several heterogeneous devices at home, the functionality to request customized video for each of them through the smartphone with preinstalled SorTube application can be added. Finally, current implementation does not deal with possible changes in network bandwidth during video streaming (Figure 6). Since video from the offloading server to the client is streamed by chunks, there is also a way for adaptive streaming realization.

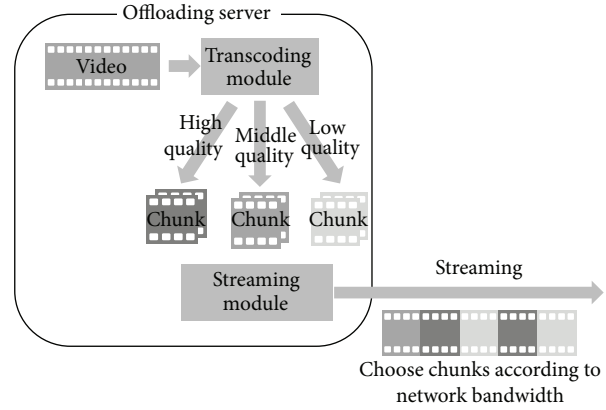

Figure 6: SorTube adaptive streaming.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was partly supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (no. R0190-15-2010, Development on the SW/HW Modules of Processor Monitor for System Intrusion Detection), the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (no. 2014R1A2A1A10051792), IDEC, the Brain Korea 21 Plus Project in 2015, Inter-University Semiconductor Research Center (ISRC), and ICT at Seoul National University and MSIP, Korea, under the ITRC (Information Technology Research Center) support program (IITP-2015-R0992-15-1006) supervised by the IITP (Institute for Information & Communications Technology Promotion).

References

[1] S. Rho, J. Cho, and E. Hwang, "Adaptive multimedia content delivery in ubiquitous environments," in *Web Information Systems Engineering—WISE 2005 Workshops: WISE 2005 International Workshops, New York, NY, USA, November 20–22, 2005. Proceedings*, vol. 3807 of *Lecture Notes in Computer Science*, pp. 43–52, Springer, Berlin, Germany, 2005.

[2] T. Repantis, Y. Drougas, and V. Kalogeraki, "Adaptive component composition and load balancing for distributed stream processing applications," *Peer-to-Peer Networking and Applications*, vol. 2, no. 1, pp. 60–74, 2009.

[3] K. Curran and S. Annesley, "Transcoding media for bandwidth constrained mobile devices," *International Journal of Network Management*, vol. 15, no. 2, pp. 75–88, 2005.

[4] A. K. W. Yim and R. Buyya, "Decentralized media streaming infrastructure (DeMSI): an adaptive and high-performance peer-to-peer content delivery network," *Journal of Systems Architecture*, vol. 52, no. 12, pp. 737–772, 2006.

[5] R. Rejaie, H. Yu, M. Handley, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications

in the internet," in *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '00)*, vol. 2, pp. 980–989, 2000.

[6] S. Dogan, A. Cellatoglu, M. Uyguroglu, A. H. Sadka, and A. M. Kondoz, "Error-resilient video transcoding for robust internetwork communications using GPRS," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 6, pp. 453–464, 2002.

[7] T. Warabino, S. Ota, D. Morikawa et al., "Video transcoding proxy for 3Gwireless mobile Internet access," *IEEE Communications Magazine*, vol. 38, no. 10, pp. 66–71, 2000.

[8] Y. Drougas, T. Repantis, and V. Kalogeraki, "Load balancing techniques for distributed stream processing applications in overlay environments," in *Proceedings of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC '06)*, pp. 33–42, IEEE, Gyeongju, Republic of Korea, April 2006.

[9] J. Noh, M. Makar, and B. Girod, "Streaming to mobile users in a peer-to-peer network," in *Proceedings of the 5th International ICST Mobile Multimedia Communications Conference (MOBIMEDIA '09)*, pp. 24:1–24:7, Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (ICST), London, UK, September 2009.

[10] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the 6th Conference on Computer Systems (EuroSys '11)*, pp. 301–314, ACM, Salzburg, Austria, April 2011.

[11] E. Cuervoy, A. Balasubramanian, D.-K. Cho et al., "MAUI: making smartphones last longer with code offload," in *Proceedings of the 8th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '10)*, pp. 49–62, San Francisco, Calif, USA, June 2010.

[12] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[13] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden, "Wishbone: profile-based partitioning for sensornet applications," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI '09)*, pp. 395–408, USENIX Association, Boston, Mass, USA, April 2009.

[14] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys '11)*, pp. 43–56, ACM, New York, NY, USA, July 2011.

[15] S. Yang, Y. Kwon, Y. Cho et al., "Fast dynamic execution offloading for efficient mobile cloud computing," in *Proceedings of the 11th IEEE International Conference on Pervasive Computing and Communications (PerCom '13)*, pp. 20–28, IEEE, San Diego, Calif, USA, March 2013.

[16] G. Kiczales, J. Lamping, A. Mendhekar et al., "Aspect-oriented programming," in *ECOOP—Object-Oriented Programming*, pp. 220–242, Springer, 1997.

[17] H.-Y. Chen, Y.-H. Lin, and C.-M. Cheng, "Coca: computation offload to clouds using aop," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '12)*, pp. 466–473, IEEE, Ottawa, Canada, May 2012.