Mobile MapReduce: Minimizing Response Time of Computing Intensive Mobile Applications

Mohammed Anowarul Hassan and Songqing Chen

Department of Computer Science,
George Mason University
{mhassanb,sqchen}@gmu.edu

Abstract. The increasing popularity of mobile devices calls for effective execution of mobile applications. A lot of research has been conducted on properly splitting and outsourcing computing intensive tasks to external resources (e.g., public clouds) by considering insufficient computing resources on mobile devices. However, little attention has been paid to the overall users' response time, where the network may dominate.
In this study, we set to investigate how to effectively minimize users' response time for mobile applications. We consider both the impact of the network and the computing itself. We first show that outsourcing to nearby residential computers may be more advantageous than public clouds for mobile applications due to network impact. Furthermore, to speed up computing, we leverage parallel processing techniques. Accordingly, we propose to build Mobile MapReduce (MMR) to effectively execute outsource computing intensive mobile applications. Based on the original MapReduce framework, a new scheduling model is built in MMR that can always leverage the best computing resources to conduct computation with appropriate parallel processing. To demonstrate the performance of MMR, we run several real-world applications, such as text searching, face detection, and image processing, on the prototype. The results show great potentials of MMR in minimizing the response time of the outsourced mobile applications.

1 Introduction

Mobile devices are getting more and more popular. According to International Data Corporation, the total number of smartphones sold in 2010 is 305 millions [5], which is a 76% increase from the previous year, and there are already over 4.6 billion mobile subscribers in the world and the number is still growing [6].

Different from traditional mobile devices (e.g., cellphones) that are mainly used for voice communication, mobile devices today are typically equipped with much more powerful processor and more sensors. Such increasing power of mobile devices has enabled fast development of mobile applications, such as picture editing, gaming, document processing, financial tracking [8]. Recently, Amazon released SDK for Android users [1] to develop mobile applications using Amazon cloud such as uploading images and videos to the Amazon cloud storage, and sharing game, movies, and other data among users.

However, constrained by the size and weight, mobile devices's processing power is still significantly lagging behind that of their desktop counterpart. Thus, many desktop applications, if running on mobile devices, can result in poor performance. For example, an OpenGL application on an Android phone can refresh slowly on the screen and drive the user away quickly. On the other hand, mobile devices are ultimately constrained by the limited battery supply and a computing-intensive application can quickly exhaust the limited battery power. Such a situation is worsened by the industrial trend to equip more sensors on mobile devices for a wider scope of non-traditional applications, such as environment monitoring [17], health monitoring [4, 7], social applications [23, 22], which are often more computing intensive.

From the resource perspective, a lots of research have considered to outsource computing intensive tasks to external resources [10, 18, 26]. For example, the virtual machine-based cloning approach [12] has been explored to clone the entire mobile environment to the cloud without worrying about modifying the application or dividing the job. Similarly, Zap takes a full process migration [25] approach with resource and process consistency. On the other hand, a number of job partitioning strategies have been proposed [28, 14] to simplify the partitioning of the existing applications between the mobile device and the external computing resources.

While many existing schemes have focused on how to split the computing-intensive tasks and outsource to external resources, the impact of the network latency on outsourced applications has not been well investigated, which may be a dominant factor in the total response time to mobile users. For mobile applications, a minimal response time is not only critical to the users' experience, but also important for preserving the limited battery power supply on mobile devices. This is particularly true for delay sensitive and interactive mobile applications. When partial tasks of such applications are outsourced, it is critical to reduce the total response time to the user in order to maintain the QoS of the application. In this paper, we aim to minimize the response time of mobile applications from the users' perspective. Since outsourcing often involves both network transferring and computing, we first show that outsourcing to appropriate resources considering data affinity and network latency could be more advantageous than public clouds in reducing the overall response time. Furthermore, to speed up computing, we leverage parallel processing techniques. Accordingly, we design and implement Mobile MapReduce (MMR) based on the original MapReduce framework. In MMR, a new scheduling model is built that can always dynamically leverage the best computing resources, be nearby computers or public clouds, with the most appropriate parallelism considering the parallelization overhead [19] for any mobile application.

To demonstrate the performance of MMR, we have built a prototype and experimented MMR with several real-world applications, including text searching, face detection, and image processing. The results show that MMR not only outperforms on-device computing by 15 times and 20 times in terms of response time and the battery power consumption, respectively, but also outperforms public cloud like Amazon EC2 by 3 times and 4 times in terms of response time and the battery power consumption, respectively.

The remainder of the paper is organized as follows. We present our motivation of leveraging residential computers and MapReduce in Section 2. We present the design of MMR in section 3. Section 4 describes mobile MapReduce implementation. We present some preliminary evaluation results with several typical applications in Section 5. Some related work is discussed in Section 6, and we make concluding remarks in Section 7.

2 Background and Motivation

In this section, we present some background and our motivation with more details.

2.1 Nearby Computers vs. Public Clouds

To study the impact of latency on outsourcing to the public clouds and nearby residential computers, we have conducted some preliminary experiments with Amazon EC2 and our local computers. We use three approaches for an experiment to find a string in a text file: 1) Local Execution in Google Nexus One with Android 2.2 OS, 1 GHz CPU, and 512 MB RAM, 2) Outsourcing to Amazon EC2 with 5 GHz CPU, 1.7 GB RAM, and 300Kbps link speed and 3) Outsourcing to Residential Computers with 2 GHz CPU, 3.2 GB of RAM, and 10 Mbps LAN. Table 1 shows the user's response time when the same program is

Table 1. Response Time (Sec)

File Size (KB)	Android	Amazon EC2	Residential Computers
10	0.0481	0.0146	0.0459
100	0.425	0.096	0.4245
200	0.424	0.971	1.300
400	0.465	1.600	1.300
750	0.480	3.400	3.300
1000	0.503	4.500	6.600

Table 2. Energy Consumption (J)

File Size (KB)	Android	Amazon EC2	Residential Computers
10	0.117	0.098	0.122
100	0.332	0.984	1.995
200	0.886	1.815	4.099
400	1.327	3.603	8.235
750	02.467	6.637	15.366
1000	3.092	8.823	20.583

executed with different approaches along with the increase of the file size. As shown in the table, although EC2 has a much faster CPU, the response time is longer than if the job is outsourced to nearby residential computers because of the impact of network latency. Correspondingly, Table 2 shows energy consumed on the mobile device for executing the program with these approaches based on Power Tutor [9].

These results show that when outsourcing mobile computing tasks, the bandwidth consumption has to be taken into consideration and sometimes this may be a dominant factor. Under such situations, outsourcing to nearby residential computers may be more beneficial than to public clouds.

On the other hand, if most of the data for the outsourced computing resides on the user's home computer, outsourcing to nearby residential computers may not outperform outsourcing to the user's home computer even if the home computer is far away. This is particularly true that today many mobile users synchronize their files with their home or office computers daily.

To demonstrate this, we experiment with compiling a Latex document with 10 files, each of 1.7 KB. In the experiment, the network speed to the farther home computer is 300 Kbps on average and to the nearby computer is 10 Mbps.

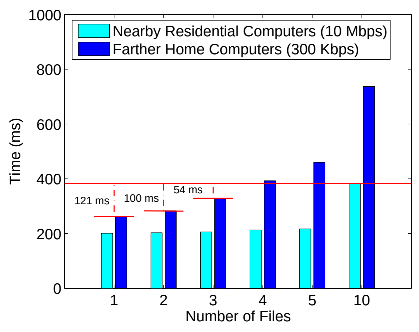


Fig. 1. Nearby Residential Computers vs. Farther Home Computer

Figure 1 shows the total execution time if 1, 2, 3, 4, 5, or 10 files needed to be transferred to nearby computers or home computers for compiling. The figure clearly shows that if the same number of files needed to be transferred, outsourcing to nearby residential computers is faster, but if the home computer only needs a small portion of the data to be transferred, then it performs better than the residential computers where the whole portion of the data needs to be sent.

The above preliminary results indicate that when outsourcing mobile applications, a scheduler should not only consider job types (such as CPU intensive or network intensive), but also the data affinity in order to minimize the impact of network.

2.2 MapReduce

As aforementioned, to speed up computing, we aim to leverage parallel processing. Since MapReduce is widely used today, we first briefly introduce the basics of MapReduce and then discuss why we adopt MapReduce.

MapReduce is a patented software framework introduced by Google to support distributed computing on large data sets on clusters of computers introduced from 2003 [16]. It is a programming model for processing and generating large data sets. Under MapReduce, the computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user specifies a `Map` function that takes an input pair to generate a set of intermediate key/value pairs. All intermediate values associated with the same intermediate key are grouped together and passed to the `Reduce` function. The `Reduce` function accepts an intermediate key and a set of values for that key. It merges together these values to form a possibly smaller set of values. In such an approach, applications are automatically parallelized and executed on a large cluster of commodity machines.

MapReduce has gained great popularity in practice. The openness of MapReduce and the simplicity of the interface allow users to develop many applications in MapReduce. Currently, there are more than 10,000 distinct programs using MapReduce[15]. MapReduce can partition the task into independent chunks and process them in parallel. In addition, the MapReduce framework deals with node failure so that re-execution of a task is minimized in case of failure. MapReduce has an open source implementation Hadoop, which we leverage in our work.

However, directly applying MapReduce for mobile computation outsourcing is not proper for a number of reasons. In the original MapReduce framework, Map and Reduce nodes are connected to each other, which is not always possible in our mobile computing environment. Furthermore, the HDFS in the original MapReduce contains the data prior to the job submission and computation, which is less likely to be practical in our mobile computing environment. That is, the data required for computing has to be transferred as well. On the other hand, the data size in our mobile computing is relatively small. It is only computing intensive relative to the slow CPU on mobile devices. Other than the above different design features, Hadoop is developed based on JVM, while Dalvik in our Android smartphone is a different platform that Hadoop is not compatible with or could be ported directly. Thus we will present later our framework that we modify from the original MapReduce in order to support our outsourcing.

3 Mobile MapReduce Design

We have shown in the previous section that nearby residential computers could outperform public clouds for some applications. Thus, from the users' perspective, it is necessary to consider all available computing resources for job outsourcing, including both residential computers and public clouds. That is, MMR should run on an architecture as shown in Figure 2.

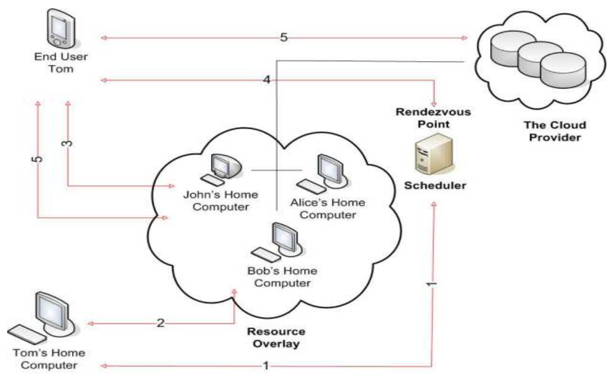


Fig. 2. Architecture of MMR

As shown in this figure, in MMR, there is a resource overlay that a participating mobile user can leverage for computing in motion. The resource overlay consists of the normal user's residential (home) computers and public clouds. A user can register her home computer in the resource overlay (**Step 2**) through the rendezvous points (**Step 1**).

When a user leaves home, she can register her home computer on the resource overlay. When the user is on the road and wants to execute some computing intensive application, she starts the job with MMR running on her mobile device. The MMR client will contact the overlay (**Step 3**) by visiting a well-known bootstrapping site. Such contact could be done through the cellular connection if WiFi is not available. In the response to this request, the user is directed to the rendezvous point (**Step 4**). Based on the geographical location of the mobile user, a list of nearby residential computers and public cloud on the overlay are identified by the rendezvous point. The MMR scheduler running on the rendezvous point further selects some of these computers based on its scheduling policy.

Once the list of residential and public cloud computers in the connectivity range is determined, the list of the machines is sent back by the rendezvous point to the mobile user (**Step 4**). From now on, all communications are done through WiFi connections. Based on the list, the mobile user connects to these computers and can start the execution. Note MMR scheduler may decide to execute part of the job on the mobile device. MMR then submits the job to an appropriate set of computers and gets back the result (**Step 5**). The MMR returns the final result to the user's application. Users' home computers operate in a P2P fashion with a credit system as proposed in [20]. With such an architecture, the most important component of MMR is the scheduler to find the most appropriate resources for outsourcing in order to minimize the response time.

In finding the most appropriate computer nodes to execute a particular task, the MMR scheduler considers the following:

1. Job Type: When requesting the computation outsourcing, the job type is determined. Based on both the CPU and the bandwidth demand and the

data locations, nearby residential computers, public cloud or the user's home computers can be selected.
2. Node Status: Since MMR mainly targets delay-sensitive applications, the status of the selected computers may affect the response time. Furthermore, it is also important for the RP to maintain load balance across different residential computers on the resource overlay.
3. Network Bandwidth: In searching for the most appropriate residential computers for outsourced computation, the computer with the highest bandwidth is always selected first if other conditions are the same.
4. Parallelization Overhead: The execution time does not decrease linearly with the increase of parallelism. In addition to the Amdahl's law, work [19] has shown that further parallelization may degrade the overall performance of the system, which is true for MapReduce as well [2]. Thus this overhead is very important in MMR.

The RP considers these factors and employs a greedy optimization to select the best strategy.

To develop our algorithm, we start from the ideal case assuming each node has equal processing power, latency, and bandwidth. Let a be the node discovery and connection set up overhead, d the time to execute the computation on a single chunk of data in MMR, and bw the time to transfer one single chunk from the mobile device to the node which depends on the bandwidth from the mobile device to all the n nodes. Note that the size of a single chunk may vary from application to application. So, if the total number of chunks is C, then the total execution time is:

$$T = a \times n + \frac{C}{n} \times d + \frac{C}{n} \times bw, \tag{1}$$

In Equation 1, we assume these nodes have the same bandwidth with the mobile device. Here the time for discovering and connecting to neighboring nodes is linearly proportional to the number of nodes as the threading capability of mobile device limited by its CPU. Differentiating Equation 1 with respect to n, we get:

$$\frac{\mathrm{d}}{\mathrm{d}n}T = a - \frac{C \times (d + bw)}{n^2} \tag{2}$$

So we get the optimal degree of parallelism when:

$$n = \sqrt{\frac{C \times (d + bw)}{a}} \tag{3}$$

In practice, residential computers may not have the same CPU, latency, and bandwidth. So a, d, and bw are not the same for each node. Moreover, some nodes may have some chunks of the input data in prior. This is similar to Multiprocessor Scheduling [13], which is NP-Complete. We propose a greedy approach to find the optimal number of nodes based on the fact that, the T in equation 1 decreases first with the increase of parallelism, then starts to increase again [19].

Algorithm 1 MMR Scheduler(I,J,L,C,cc)

```
1: $LC \leftarrow$ The set of available computers nearby from I and L
2: **for** each node $i$ in $LC$ **do**
3:     add $i$ to $LC$
4:     $i_a \leftarrow node\_connection\_overhead_i$
5:     $i_d \leftarrow cc/i_c$
6: **end for**
7: $n \leftarrow \|LC\|$
8: Calculate $a_t, d_t$, and $bw_t$: the average of $i_a, i_d$, and $i_{bw}$ of all nodes $i$ in $LC$
9: $op\_n \leftarrow \sqrt{\frac{C\times(b_t+bw_t)}{a_t}}$
10: **for** each $i$ in $LC$ **do**
11:     $Weight_i \leftarrow \frac{1}{i_a+i_d\times\frac{C}{op\_n}+i_{bw}\times\frac{C}{op\_n}}$
12: **end for**
13: Sort $LC$ in decreasing order according to $Weight_i$
14: $a_{avg} \leftarrow i_a$ of first node of $LC$
15: $d_{avg} \leftarrow i_d$ of first node of $LC$
16: $bw_{avg} \leftarrow i_{bw}$ of first node of $LC$
17: $totaltime \leftarrow a_{avg} + C \times d_{avg} + C \times i_{bw}$
18: $count \leftarrow 1$
19: **for** $i = 2$ to $n$ **do**
20:     $a_{avg} \leftarrow$ average of first $i$ nodes' $i_a$
21:     $d_{avg} \leftarrow$ average of first $i$ nodes' $i_d$
22:     $bw_{avg} \leftarrow$ average of first $i$ nodes' $i_{bw}$
23:     $temptotaltime \leftarrow a_{avg} \times i + \frac{(C\times d_{avg})}{i} + \frac{C\times bw_{avg}}{i}$
24:     **if** $temptotaltime \leq totaltime$ **then**
25:         $totaltime \leftarrow temptotaltime$
26:         $count \leftarrow count + 1$
27:     **end if**
28: **end for**
29: **for** $i = 1$ to $n$ **do**
30:     $C_i \leftarrow \frac{Weight_i}{\sum_{\forall i\in LC} Weight_i} \times C$
31: **end for**
32: Let $LCRD$ be the set of nodes having portion of the input data in prior
33: $m \leftarrow \|LCRD\|$
34: Sort LCRD according to $i_{ds}$ in decreasing order
35: $j \leftarrow 1$
36: **for** $i = 1$ to $m$ **do**
37:     **if** $j \leq count$ **then**
38:         $Time_i \leftarrow i_a + \frac{C_j - i_{ds}}{i_{bw}} + C_j \times i_d$
39:         $Time_j \leftarrow j_a + \frac{C_j - j_{ds}}{j_{bw}} + C_j \times j_d$
40:         **if** $Time_i < Time_j$ **then**
41:             First remove $i$ from $LCRD$ and then Insert node $i$ in between $j$ and $j-1$
                in $LC$
42:         **else**
43:             $j \leftarrow j + 1$
44:         **end if**
45:     **end if**
46: **end for**
47: **return** first $count$ nodes from $LC$
```

Algorithm 1 shows the pseudo-code of the algorithm. In the MMR Scheduler, `I` represents the user's `id`, `L` is the `location`, which is required to find the nearby computers available for computation outsourcing. `J` represents the job. `LC` is the list of the available computers, which holds the following information for each node on the list: 1) i_c: computation power available on node i; 2) i_l: location of node i; 3) i_{bw}: the time to transfer one chunk of data from the mobile device to the node depending on the channel capacity from the mobile device to the node i; 4) i_{ds}: chunks of the data stored in prior for a job J in the node i; 5) C: total size of the input chunks of the input data; 6) cc: CPU cycle required for execution of each input chunks. Note that we assume profiling can be used to obtain these parameters for mobile applications.

Note that we include the mobile device and the home computer of the user as potential candidates to execute computation, as computation on them may be economical. As shown in the algorithm MMR Scheduler, in the first step, MMR finds the set of reachable computers nearby LC based on the location of the nodes and the mobile device (Line 1). We calculate a and d for each node, (Line 2-6) and the average of them (Line 8). Then we assign weight to different node based on the heuristics (Line 10-11). We first assume that we are in the ideal case where all nodes in LC have the same a, d, and bw: a_t, d_t and bw_t namely which are the average of those values for all nodes in LC. We also assume to use an optimal number of nodes if all of them have their average a, d, and bw. So, to process equal number of chunks $\frac{C}{op\_n}$, the time taken by each node is $i_a + i_d \times \frac{C}{op\_n} + i_{bw} \times \frac{C}{op\_n}$. where op_n is the optimal number of nodes according to Equation 3 (Line 9). The *Weight* of each node is the inverse of the time taken by each node to get and process the $\frac{C}{op\_n}$ number of chunks. Note that the more powerful and the shorter the latency a node has, the lower the time and the higher the *Weight*.

We sort the LC in decreasing order of weight (Line 13). Then we could find the minima of the curvature of the T in Equation 1 by adding one by one node for our computation (Line 19-28). Here we assume the a, d, and bw be the average of those values of the total nodes considered so far.

We also calculate the potential number of chunks to be executed by each node here based on the weight heuristic (Line 30). Then we find out the nodes having some chunks of the data in prior and sort them in decreasing order of i_{ds}(Line 34) in list $LCRD$ and give them priority if they have better performance considering affinity data (Line 36-46). The key point here is that, we first take the first node in $LCRD$ and take the first node of sorted LC and compare the total data transfer and execution time of the two nodes with the associated number of chunks of the node in LC and local data chunks. If the node from $LCRD$ takes less time than the node of LC, we move that node of $LCRD$ forward in the list LC before the current node and deal with the next node of $LCRD$ in the same way. Otherwise, we compare the node of $LCRD$ with the second node of LC in the same way until the list $LCRD$ is visited or we have taken the first *count* nodes from LC. In this way, we get the first *count* nodes having optimal parallelization and performance with respect to latency, computation power, and data location. The complexity of this algorithm is $\theta(n)$ for the first four loops, $\theta(n \log n)$ for the sorting, and $\theta(n + m)$ or $\theta(n)$ as $m < n$ for the nested loop (Line 36-46). The above algorithm deals with how to select the most appropriate computing resources for one job. If there are multiple jobs, MMR follows the original Hadoop fair scheduling policy for the queued jobs and then the MMR scheduler selects the best nodes for each job.

4 Mobile MapReduce Implementation

A major component of MMR is the modified MapReduce that considers user mobility and unreliability of residential computers. We modify the original MapReduce to incorporate these new functions as follows.

4.1 MMR vs. Hadoop

Our MMR implementation is based on the widely used open source Hadoop. From the perspective of the framework, MMR differs from the original Hadoop in the following aspects so that we need to modify the original Hadoop to accommodate these.

- *Dynamic Mobility Property of MMR:* In Hadoop, the master node has the worker list beforehand. It configures the network first before submitting any job. The worker nodes are responsible themselves to join the network. But in MMR we propose the framework for mobile users without persistent connections and the master node does not have any knowledge about the neighboring worker nodes.
- *Non-Distributive File System of MMR:* In Hadoop, HDFS [27] is used to store the input file. In MMR, the selected nearby residential computers do not have a copy of the input file until the file is transferred there. Thus, upon the response from the RP, the input data is first split and transferred to the selected worker node before the computation begins.
- *Handling Isolated Worker Nodes:* In MapReduce, the intermediate <Key,Value> pairs produced by the Map nodes are periodically written to local disk. The locations of these buffered key/value pairs on the local disk are passed to the master who forwards these locations to the Reduce workers. The Reducers use remote procedure calls to read the buffered data from the local disks of the map workers.
  In our resource overlay network, it is highly possible that the Map/Reduce nodes are not directly connected. So Reduce nodes cannot execute RPC to fetch the buffered data. In our framework, the Map node sends the list of <Key,Value> pairs to the master node who eventually forwards to the Reducers.
- *Node Failure:* MMR detects the failure of worker nodes with a timeout mechanism. Whenever the master node submits a job to execute to any worker node, it periodically contacts the worker node. The master node also gets the result

back for small chunks of the job finished in that worker and saves it. Note that the worker node also gives a backup copy to the resource overlay. If it is the master node that loses the connection, then the resource overlay may give the result back to the master node based on the backup copy. If it is a crash of the worker node, then the mobile node finds another worker if any, or it executes the job locally (on the mobile device). After finding another machine (worker or local), it executes the remaining portion of the job. As the input data is partitioned into small independent chunks, the failure of any worker causes only re-execution of that portion of data.
– *Minimizing Intermediate <Key, Value> Transfer Overhead:* To minimize the transferring overhead, in MMR, the *MMR Combiner* is run over the <Key, Value> pairs produced by the Map nodes. For example: suppose we are searching for a rare string "MMR" in a large file. In the original MapReduce, the Map will emit <"MMR", "one"> every time it finds that string in the file. The Reducers would fetch the intermediate key/value pairs and merge them. In MMR, to reduce the data communication overhead, we propose that the *MMR Combiner* would integrate all the occurrence and finally send the result when it is finished for a particular split of input. If there are ten "MMR" strings in the input split of the file, the node emits one < "MMR", "ten"> pair instead of sending ten < "MMR", "one"> pairs. This change causes significant improvement for data communication.

In addition, as the result of mobile application is comparatively small, so it is much faster to keep it in the cache rather than to write in local file system of the Workers. We also find that the system performs better when there is no replication factor like original MapReduce as the node failure is common in MMR. MMR uses a smaller block size, such as 512 KB, which is smaller than the original Hadoop (64 MB) due to the nature of the jobs on mobile devices. In practice, the block size can also be determined dynamically based on the input size, the reliability of nearby computers, the available network bandwidth, etc.

4.2 MMR Workflow

With the above modifications from the original Hadoop, in MMR, the mobile device works as the master node and the residential computers (including the home computer of the user in case) works as worker nodes. Each worker node may work as Map or Reduce. Figure 3 illustrates the similarities and the modification between MapReduce and MMR.

After exploring neighboring residential computers, the MMR Scheduler sends the list of the residential computers to the MMR running in mobile device. **(Step 1)**. MMR establishes connections between the master node running in mobile device and the worker nodes running in the residential computers. The nodes may establish authentication and trust for this purpose. The name node and the job tracker of MapReduce starts on the master node. The name node divides the whole input data into `n` small chunks and read input chunks **(Step 2)**. It then sends the data and the computation to the appropriate data nodes **(Step 3)**.

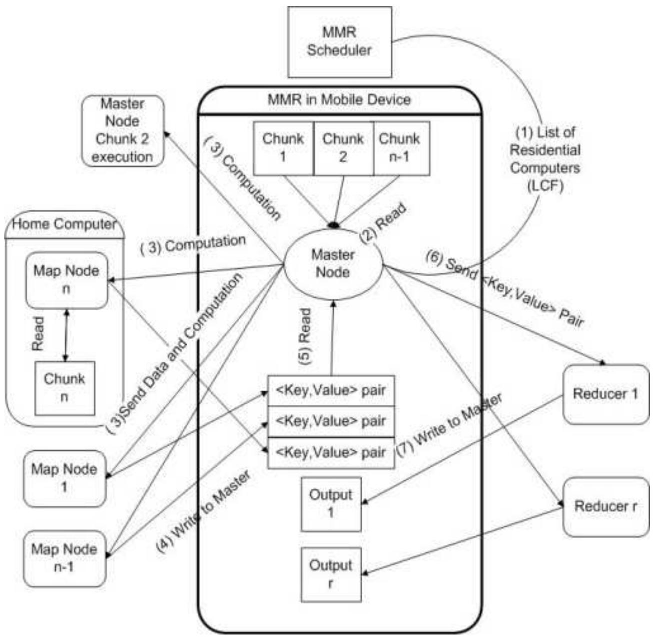


Fig. 3. MapReduce and MMR

Note here that some data nodes may have some data in prior. These nodes are referred by the MMR scheduler so that only the computing program needs to be sent there, thus reducing the data transferring overhead. Some data chunks may be executed in the mobile device itself due to security reason. For all the other nodes, both computing program and data need to be sent over. The job tracker in the mobile device keeps track of the the processing progress of the job.

Currently we send the computation in serializable byte code from the master node to the data nodes. It also defines the output class key/value as well.

With the above interface, application developers can define the class of map, key/value and the input/output format. Thus the worker nodes do not have to know about the on-going computations and can offer unlimited types of services with its simple model without reconfiguring itself.

The data node receives the data portion in desired input data format and the task tracker starts the computation and sends back the output to the job tracker **(Step 4)**.

The job tracker saves the intermediate key/value results and marks the entry for that particular portion of the input data chunk as finished. The name node sends the next portion of the job to the data node and the task tracker starts computation for the next portion of the job.

This continues until the job is finished or any data node fails (e.g., a node is shut down or a node loses the connection from the name node). If any worker node fails, MMR tries to get the result for that data portion from the resource overlay or re-execute that portion of the job by exploring another worker node.

After finishing the Map part, the reducer may begin either on the mobile device or again in the nearby worker node acting as Reducer depending on the volume of computation. The master node reads the intermediate <Key,Value> pairs **(Step 5)** and sends them to the reduce nodes **(Step 6)**. After the computation is finished, reduce node sends the result back **(Step 7)**.

5 Preliminary Evaluation

5.1 Experiment Setup

In the experiments, we emulate a 3 user model and they have identical residential computers and mobile devices. We use Google Android Nexus One with 1 GHz CPU and 512 RAM as the mobile device for the local execution. We use 5 lab computers to emulate overlay residential computers that all have a dual-core CPU with 2GHz and 2 GB RAM for outsourcing and parallelization in residential computers. The EC2 instances rented are at Northern Virginia Data Center of Amazon. Each remote EC2 Ubuntu instance has a 5 GHz CPU with 1.7 GB of RAM. We use Power Tutor [9] to measure the power consumed by the applications running on the smartphone. The WiFi is 10 Mbps and the bandwidth from the Android Phone to EC2 instance is around 300 Kpbs on average. Note that we have also experimented with different network speeds. We omit their results for brevity.

We have conducted experiments with the following three applications.

– *Text Search* In this application, the user searches a string in a text file and the frequency of occurrence of that string is returned to the user. This simple string counting application takes an input file of 2.6 MB. We use string matching to find the total number of occurrence of that string in that text file.
– *Face Detection* In this application, we take a picture of a human face and try to match it with all the pictures in a folder previously taken. We use Cross-Correlation Function [3] to find the correlation between an image pair. Based upon that, we detect a particular person. We have each image in a different jpg file. The correlation between the files has been calculated by taking input from three different streams for 3 RGB values. The resultant size for the reference images is 575 KB in total and the newly taken image size is 145 KB. So the total size of the data file is 720 KB, and the computation program is 3 KB.
– *Image Sub Pattern Search* In this application, we take a picture and try to find the picture as a part of another large picture in a folder previously taken. We use Cross-Correlation Function [3] and 2D Logarithmic Search [21] to find the sub-image. We have each image in a different jpg file. The correlation between the files has been calculated by taking input from three different streams for 3 RGB values. The resultant size for the reference image is 1.7 MB, the newly taken image size is 260 KB, and the computation program is 4 KB.

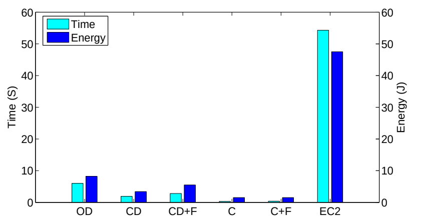
We profile each application to deduce the average CPU cycle and data transfer requirement. We fed these profiling results and locations for the MMR Scheduler to find nearby residential computers to outsource computation. While profiling, we assumed that either there is no data is stored in prior in the residential computers or all the data are stored there. Upon this experimental set up we run each application in the following different environments.

– On-device*(OD)*: We run the application on the mobile device directly here.
– Computation+Data*(CD)*: Both the computation program and the data are outsourced to the residential computers here. The MMR in the mobile device gets connected to the resource overlay and the rendezvous point to explore the neighboring residential computers and outsource the computation with the data file.
– Computation+Data+Node Failure*(CD+F)*: This is to consider the node failure in the above environment to study the impact of node failure. When a node fails, MMR contact the resource overlay and the rendezvous point once again to find another neighboring residential computer and start the job from the failed point by outsourcing the data from the failed portion and the original computation program.
  To emulate node failure case, we deliberately turn off one computer in the middle of an on-going computation when the computation is about 50% completed. Then MMR detects the failure based on timeout and it contacts resource overlay and rendezvous point to find another nearby residential computer.
– Computation*(C)*: We emulate the scenario when the selected residential computer is the user's home computer, which has the data of the task. The mobile device only needs to transmit the computation for the applications, which is small in size.
– Computation+Node Failure*(C+F)*: This is to consider the node failure in the above environment. Note that here for both the failed node and the new node, we outsource only the computation. We emulate the node failure case as we have done for CD+F.
– EC2 *(EC2)*: We outsource the computation to the remote amazon EC2 instances. We assume that EC2 is always available and 100% reliable.

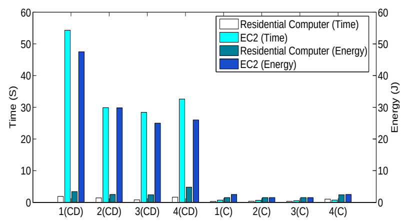
In all these experiments, we mainly focus on the response time and the energy consumption on the mobile device. Since we use homogeneous machines for both residential computers and Amazon EC2 instances, our scheduler follows Equation 3. We further test with other parallelism levels in order to compare their performance.

5.2 Experimental Results

In this section, we describe the performance of the different approaches we have tested for the three applications. We repeat each experiment five times and present the average of the results.



(a) Response Time and Energy Consumption



(b) Response Time and Energy Consumption for Parallel Processing

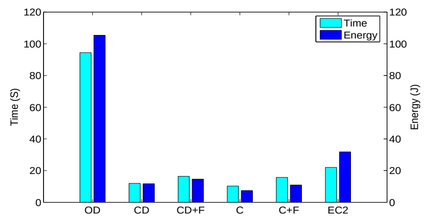
Fig. 4. Text Search

Text Search Figure 4 depicts the response time and the energy consumption of the text search application when it is executed on the Android and the computation is outsourced with and without parallel processing. In this experiment, the amount of data transferred is 2.6 MB when both computation and data are transferred, otherwise it is 1 KB if only the computing program is needed to be outsourced. In outsourcing the data file, it is divided into 512 KB chunks. So in the cases with node failure (`CD+F` and `C+F`), only one chunk is lost, which minimize the re-execution overhead.

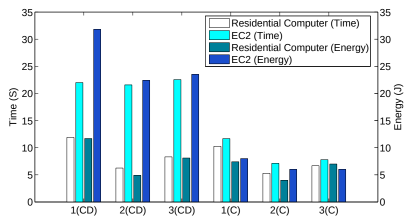
In Figure 4, the left-y axis represents the response time while the right-y axis represents the corresponding energy consumption. Figure 4(a) clearly shows that outsourcing to EC2 results in the worst performance in terms of both the response time to the user and the amount of energy consumed, even worse than the on-device execution. Compared to the case when it is executed locally on the mobile device, outsourcing to the nearby residential computers results in 69% and 59% less response time and energy consumption, respectively, although outsourcing to nearby residential computers demand some bandwidth for file transferring. In the node failure cases where residential computers may not be reliable or the home user can depart a residential computer from MMR at any time, Figure 4(a) shows that the performance of outsourcing still outperforms the on-device computing in terms of both the response time and the total energy consumed on the mobile device, although there is a 47% and 61% increase compared to if there is no node failure.

When the computation is parallelized among multiple machines, Figure 4(b) shows the result. Again, the left-y axis represents the response time while the right one represents the energy consumption. The residential computers are identical with a 2 GHz CPU and 2 GB RAM. The rented EC2 instances have 5 GHz CPU and 1.7 GB RAM each. Without parallel processing, the response time may be well over the average connection time of a mobile user with a roadside WiFi ranges between 6-12 seconds [24]. This makes it impossible for a mobile user to get the result in time in the same communication session although EC2 has a faster CPU speed. This would be a critical problem for delay sensitive mobile applications when a user waits to get the result back. As shown in the figure, parallelization can clearly improve the performance when the number of computers is increased from 1 to 2 and 3. However, Figure 4(b) also shows that the response time and energy consumption first decrease with the increase of parallelization level, then it increases when the parallelization level increases (from 3 to 4 nodes). So here it is also important to calculate the the appropriate degree of parallelism to optimize the performance.

Again, in Figure 4(b), we also observe that the residential computers perform significantly better than EC2 when both the data and computation (`CD`) are outsourced. But when only the computation (`C`) is outsourced, they have similar performance.



(a) Response Time and Energy Consumption



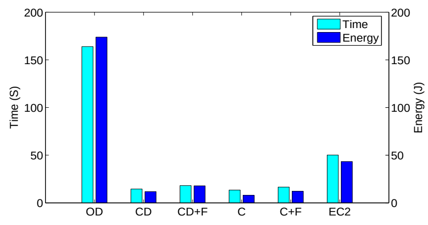
(b) Response Time and Energy Consumption for Parallel Processing

Fig. 5. Face Detection

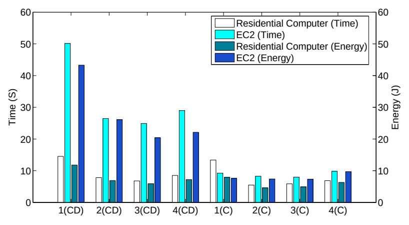
Face Detection Figure 5 shows the performance results when the face detection program is executed in different environments. In particular, figure 5(a) shows that executing on the Android takes the longest time of about 94.5 seconds. Not surprisingly, the corresponding energy consumption is the largest for the on-device execution.

Figure 5(a) also shows that both the response time and the energy consumption are reduced when the computation is outsourced. When the program is outsourced to the nearby residential computers, the performance improvement is more pronounced than when the program is outsourced to EC2: on the residential computer, the response time is about 10.25 seconds and 11.90 seconds without or with the data transferred. Correspondingly, when the computation is outsourced to the nearby residential computer, the energy consumed is only about 23% and 36%, respectively, of the total energy consumption when we have on device execution without or with data transfer.

With the help of parallelization, the performance is better. Figure 5(b) shows the effect of parallelism on the response time and the energy consumption. However, as shown in the figure, although using 2 nodes to parallelize the computation does improve the user response time and the total energy consumption on the mobile device, the response time and energy consumption of the computation actually increase when the parallelization is further increased (from 2 nodes to 3 nodes). When the computing nodes have the data in prior, the performance is better than when the data need to be actually transferred before the computation. This indicates outsourcing computation to the nodes where data resides may be more beneficial than to the nodes with higher computation power without any data in prior. But again, an appropriate parallelization level is always desired as more resources may not improve the performance.



(a) Response Time and Energy Consumption



(b) Response Time and Energy Consumption for Parallel Processing

Fig. 6. Image Sub Pattern Search

Image Sub Pattern Search For *Image Sub Pattern Search*, Figure 6(a) shows that executing on the Android takes the longest time of about 163.9 seconds. In all the outsourcing scenarios, the response time is significantly reduced. Correspondingly, the energy consumption is the largest for the on-device execution. The reduction when the program is outsourced to the nearby residential computers is more pronounced than when the program is outsourced to EC2: on the residential computer, the response time is about 13.37 seconds and 14.52 seconds without or with the data transferred. Correspondingly, the energy consumed is only about 10% and 11% of the On-device (OD) computation, respectively, when the computation is outsourced to nearby residential computers. The node failure only causes 54% and 51% increase of the response time and the energy consumption compared to their counterpart without any node failure. These results are still much better than those when outsourcing to EC2.

Figure 6(b) shows the response time and the energy consumption when the computation is parallelized. When two and three nodes are used for computing, both the response time and the energy consumption on the mobile device decrease. However, when more nodes are used in the computing, performance degrades due to parallelization overhead. On the other hand without parallelization, the response time is more than 10 seconds, but with parallelization, the user gets the result back in 5 seconds, which is more feasible.

The experimental results show that while computation is outsourced to residential computers, the overall performance is better than when the computation is outsourced to EC2, though EC2 is much more powerful than nearby residential computers in terms of the CPU speed. The performance can be further improved when the computation is parallelized. The average gain for the response time and energy consumption is about 1.5 to 2 times compared to its single node computation on average. However, with parallel processing, appropriate parallelization is desired. This is because a certain level of parallelization can help reduce the response time and the total energy consumption on the mobile device, and further increasing parallelization level may actually degrade the performance. This is due to the fact that parallelization involves overhead, which may dominate under certain circumstances. Thus, in scheduling the execution of outsourced tasks, this must be taken into consideration as we have done in MMR scheduler.

6 Related Work

Plenty of research has been conducted to outsource computing tasks to external computing sources [10, 18, 26, 11]. Typically, these schemes focus on how to properly split the job and deploy on the external computing sources. For example, studies [28, 12] demonstrate the ability to partition the application and associate classes and thus outsourcing them. Rudenko et al. [26] suggest that if the total energy cost of sending the task else where and receiving the result back is lower than the cost of running it locally, then remote process execution can save battery power. Flinn et al. [18] also propose a similar idea, in which remote execution simultaneously leverages the mobility of mobile devices and the richer resources of large devices. Balan et al. [10, 11] propose to augment the computation and storage capabilities of mobile devices by exploiting the nearby (surrogate) computers. Recently, MAUI [14] is proposed to partition the program dynamically and submit it on surrogate computers.

However, existing work has considered little about how to minimize the response time of the outsourced application, where the network latency may dominate. Considering network transferring and appropriate parallel processing, MMR partitions the data into small chunks to keep the re-execution tractable. Compared to MAUI [14] that requires modification of each application, MMR aims to work transparently with the existing over 10,000 programs developed based on MapReduce. In addition, compared with the partitioning overhead of existing approaches [28, 14], the simple two methods Map and Reduce inherited from MapReduce offer an simple interface to be implemented in practice.

7 Conclusion

While mobile devices and mobile applications are getting more and more popular, effectively and properly executing these mobile applications is challenging. In this work, we focus on how to minimize the users' response time of these applications from the users's perspective. Considering both the network impact and the computing itself, we have designed and implemented Mobile MapReduce based on the original MapReduce framework for this purpose. Experimented on a prototype, Mobile MapReduce demonstrates that it can effectively minimize user's response time.

Acknowledgement

We appreciate constructive comments from anonymous referees. The work is partially supported by NSF under grants CNS-0746649 and CNS-1117300 and AFOSR under grant FA9550-09-1-0071.

References

1. AWS SDK for Android. http://aws.amazon.com/sdkforandroid/.
2. BlastReduce: High Performance Short Read Mapping with MapReduce. www.cbcb.umd.edu/software/blastreduce/.
3. Cross Correlation . http://en.wikipedia.org/wiki/Cross-correlation.
4. Diamedic. Diabetes Glucose Monitoring Logbook. http://ziyang.eecs.umich.edu/projects/powertutor/index.html.
5. International Data Corporation : Press Release 28 Jan and 4 Feb, 2010. http://www.idc.com/.
6. International Telecommunication Union : Press Release 10 June, 2009. www.itu.int.
7. iPhone Heart Monitor Tracks Your Heartbeat Unless You Are Dead. gizmodo.com/5056167/.
8. Mint. http://www.mint.com/.
9. Power Tutor. http://ziyang.eecs.umich.edu/projects/powertutor/index.html.
10. Rajesh Balan, Jason Flinn, M. Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. The case of cyber foraging. In *Proceedings of the 10th ACM SIGOPS European Workshop*, Saint-Emilion, France, July 2002.
11. Rajesh Krishna Balan, Darren Gergle, Mahadev Satyanarayanan, and James Herbsleb. Simplifying cyber foraging for mobile devices. In *Proceedings of The 5th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, San Juan, Puerto Rico, June 2007.
12. Byung Gon Chun and Petros Maniatis. Augmented smartphone applications through clone cloud execution. In *Proceedings of the 12th Workshop on Hot Topics in Operating Systems (HotOS)*, Monte Verit, Switzerland, May 2009.
13. Pierluigi Crescenzi and Viggo Kann. A compendium of NP optimization problems. 1998.
14. Eduardo Cuervo, Aruna Balasubramanian, Dae ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. MAUI: Making smartphones last longer with code offload. In *Proceedings of The 8th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, San Francisco, CA, USA, June 2010.
15. Jeffrey Dean and Sanjay Ghemaawat. Mapreduce a flexible data processing tool. In *Communication of the ACM*, Jan 2010.
16. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI)*, San Francisco, CA, Dec 2004.
17. Jakob Eriksson, Lewis Girod, Bret Hull, Ryan Newton, Samuel Madden, and Hari Balakrishnan. The pothole patrol: Using a mobile sensor network for road surface monitoring. In *Proceedings of The 6th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Breckenridge, Colorado, June 2008.

18. Jason Flinn, Dushyanth Narayanan, and M. Satyanarayanan. Self-tuned remote execution for pervasive computing. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, Schloss Elmau, Germany, May 2001.
19. Johnson M. Hart. Data processing: Parallelism and performance. In *MSDN Magazine*, Jan 2011.
20. Mohammed Anowarul Hassan and Songqing Chen. An investigation of different computing sources for mobile application outsourcing on the road. In *Proceedings of the 4th International ICST Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications (Mobilware)*, June 2011.
21. J. R. Jain and A. K. Jain. Displacement measurement and its application in interframe image coding. In *IEEE Transactions on Communications*, volume 29, December 1981.
22. Seungwoo Kang, Jinwon Lee, Hyukjae Jang, Hyonik Lee, Youngki Lee, Souneil Park, Taiwoo Park, and Junehwa Song. Seemon: Scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In *Proceedings of The 6th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Breckenridge, Colorado, June 2008.
23. Bin Liu, Peter Terlecky, Amotz Bar-Noy, Ramesh Govindan, and Michael J. Neely. Optimizing information credibility in social swarming applications. In *Proceedings of IEEE InfoCom, 2011 mini-conference*, Shanghai, China, April 2011.
24. Jőrg Ott and Dirk Kutscher. Drive-thru internet: IEEE 802.11b for Automobile Users. In *Proceedings of IEEE InfoCom*, Hong Kong, March 2004.
25. Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh. The design and implementation of zap: A system for migrating computing environments. In *Proceedings of the 5th Symposium on Operating System Design and Implementation (OSDI)*, Boston, MA, Dec 2002.
26. Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. In *Proceedings of Mobile Computing and Communication Review (MC2R)*, 1998.
27. Tom White. Hadoop: The definitive guide.
28. K. Nahrstedt X. Gu, A. Messer, I. Greenberg, and D. Milojicic. Adaptive offloading inference for delivering applications in pervasive computing environments. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications(PerCom)*, Dallas-Fort Worth, Texas, March 2003.