A Framework for Partitioning and Execution of Data Stream Applications in Mobile Cloud Computing

Lei Yang, Jiannong Cao, Yin Yuan, Tao Li, Andy Han, and Alvin Chan
Department of Computing, Hong Kong Polytechnical University, Hong Kong

ABSTRACT
The contribution of cloud computing and mobile computing technologies lead to the newly emerging mobile cloud computing paradigm. Three major approaches have been proposed for mobile cloud applications: 1) extending the access to cloud services to mobile devices; 2) enabling mobile devices to work collaboratively as cloud resource providers; 3) augmenting the execution of mobile applications on portable devices using cloud resources. In this paper, we focus on the third approach in supporting mobile data stream applications. More specifically, we study how to optimize the computation partitioning of a data stream application between mobile and cloud to achieve maximum speed/throughput in processing the streaming data.

To the best of our knowledge, it is the first work to study the partitioning problem for mobile data stream applications, where the optimization is placed on achieving high throughput of processing the streaming data rather than minimizing the makespan of executions as in other applications. We first propose a framework to provide runtime support for the dynamic computation partitioning and execution of the application. Different from existing works, the framework not only allows the dynamic partitioning for a single user but also supports the sharing of computation instances among multiple users in the cloud to achieve efficient utilization of the underlying cloud resources. Meanwhile, the framework has better scalability because it is designed on the elastic cloud fabrics. Based on the framework, we design a genetic algorithm for optimal computation partition. Both numerical evaluation and real world experiment have been performed, and the results show that the partitioned application can achieve at least two times better performance in terms of throughput than the application without partitioning.

Keywords
mobile cloud computing; application partitioning; genetic algorithm;

1. INTRODUCTION
Cloud computing is an important paradigm shift in IT service delivery driven by economies of scale. It provides a computing paradigm that enables a shared pool of virtualized, dynamically configurable, and managed computing resources to be delivered on demand to customers over the Internet and other available networks. On the other hand, with the advances in technologies of wireless communications and portable devices, mobile computing has become integrated into the fabric of our every day life. With increased mobility, users need to run stand-alone and/or to access remote mobile applications on mobile devices.

The application of cloud services in the mobile ecosystem enables a newly emerging mobile computing paradigm, namely Mobile Cloud Computing (MCC). MCC offers great opportunities for mobile service industry, allowing mobile devices to utilize the elastic resources offered by the cloud. There are three MCC approaches: 1) extending the access to cloud services to mobile devices; 2) enabling mobile devices to work collaboratively as cloud resource providers; 3) augmenting the execution of mobile applications using cloud resources, e.g. by offloading resource intensive computing tasks on mobile devices to the cloud. This will allow us to create applications that far exceed traditional mobile device’s processing capabilities.

In the first approach, users use mobile devices, often through web browsers, to access software / applications as services offered by cloud. The mobile cloud is most often viewed as a Software-as-a-Service (SaaS) cloud. All the computation and data handling are usually performed in the cloud. The second MCC approach makes use of the resource at individual mobile devices to provide a virtual mobile cloud, which is useful in an ad hoc networking environment without the access to the Internet cloud. The third MCC approach uses the cloud storage and processing for applications running on mobile devices. The mobile cloud is considered as an Infrastructure-as-a-Service (IaaS) or Platform-as-a-Service (PaaS) cloud, which is leveraged to augment the capability of mobile devices through partial or full offloading of the computation and data storage from the mobile devices.

In this paper we consider the third MMC approach. By moving the computation to the cloud, many applications which could not be accommodated before due to the lack of significant computation capability and energy power of mobile devices, will be made possible, while leveraging the stable and ample capacity of cloud. We focus on one class of these applications, namely mobile data stream applications. These applications usually use camera and/or other high data rate sensors to perform perception related tasks, like face or object recognition, to enable augmented-reality experiences on mobile devices. Specifically, these applications have two characteristics. First, they require continuous processing of high data rate sensors such as camera to
maintain the accuracy. For example, a low frame rate may miss intermediate object poses or human gestures. Second, the computer vision and machine learning algorithms used to process streaming data are often computation-intensive.

However, to make sure that the MCC approach can really bring benefits to both the end users and application providers, we need to address the following two problems. (1) **Application partitioning problem:** given a mobile application which consists of a set of computational tasks, which tasks should be offloaded to the clouds such that the mobile end users could experience the **maximal performance**? For data stream application, one important performance metric is the speed/throughput that the application processes the streaming data. (2) **Load scheduling problem:** for the cloud-scale applications, it is possible that a large number of mobile users offload the computational tasks onto cloud at the same time. So how to schedule the offloaded tasks in the cloud such that the **utilization of cloud resources is minimized**? It is critical for the application provider to save their operational cost.

Although the two problems are posed respectively from the requirements of the end users and the application provider, they need to be solved within one system framework. So far, there is no work that treats these two problems jointly. Existing systems [4][1][15][6][12] that support the application partitioning are only suitable for traditional mobile Internet computing, and do not give any solution on how to use the resources efficiently in clouds to make the applications scalable in cases of serving a large number of mobile users. Other efforts [2][10][13][7][11] in facilitating large scale cloud applications do not fit well in the MCC applications because they do not support adaptive partitioning of the application between the client and clouds.

In this paper, we propose a framework for partitioning and execution of the data stream applications under the third MCC approach. The framework contains a novel system architecture and algorithm which solves the fore-mentioned problem, aiming at achieving maximal performance experienced by the end users and minimal cost in cloud resources favored by the mobile application providers. The main contributions of this paper are described as follows:

- **We design a system architecture** for the advanced MCC applications. The design is placed on existing mobile platforms and cloud fabrics. The architecture contains two critical mechanisms. First, through online profiling of the characteristics of mobile devices and wireless bandwidth at mobile side, and the back-up of the partitioning results at cloud side, a mechanism is designed to enable fast and adaptive partitioning of the application. Second, a multi-tenancy mechanism is adopted in clouds so that the offloaded computational instances can be shared by multiple mobile users in case they offload the same tasks onto clouds.

- **We propose an optimal partitioning algorithm** for mobile data stream applications. To the best of our knowledge, it is the first work to study the partitioning problem for data stream applications which require parallel execution of different operations onto the streaming data to achieve high processing speed.

- **We demonstrate the efficiency of the proposed algorithm** through extensive simulations. More importantly, we develop a representative, real world application namely QR-code recognition, and validate the effectiveness of our design through experimental tests on the application.

In the remaining part of the paper, we first describe the applications and design requirements in Section II. We then present the architectural design in Section III. The application partitioning problem and solutions are presented with details in Section IV. Numerical evaluation and real world experiment are presented in Section V and Section VI. Section VII describe the related works, and Section VIII concludes the paper.

2. BACKGROUND

Before describing the design of the MCC system architecture, we first discuss the application that the system can support. The infrastructures that the system is designed based on are then presented. We also point out the design requirements and key methods.

2.1 Mobile Data Stream Applications

Our system targets for the mobile data streaming applications. These applications take the streaming data as input, perform a series of operations onto the data, and then output the results. The input data are sampled periodically from the sensors on the mobile device. Mobile augmented reality is considered as one killer streaming application. The application use the camera and/or other sensors to percept
the user’s environment/scene, and then augment the original scene with relevant information in the display. The perception is done continuously. The core part of AR applications is the image based object recognition. Fig.1 shows the operations involved in the whole process of imaged based object recognition. Note that the SIFT algorithm is used to extract the features [24].

We use a dataflow graph to model the data stream application. The dataflow graph is composed of a set of components and a set of channels as shown in Fig.2. The components run concurrently with each one performing its own functional operations onto the data. The component has input ports and output ports. Each port is associated with a specific data type. The channel’s capacity is defined as the maximum number of units of data the channel is able to hold. The channel also indicates the precedence constraint between the operations/components for processing one unit of data, which means the component can not process the data until all of its precedent components complete the operation on that data. The component processing the input data of the application is called the entry node. The component generating the output data is called the exit node. In real implementations, the components are mapped into threads or processes. The channels are usually implemented by means of TCP sockets, or shared memory or persistent storage. The dataflow model is based on a data centric approach and usually takes advantage of pipeline to accelerate data processing.

2.2 System Model

The MCC system model consists of three parts: mobile clients (devices), wireless networks and the cloud (data centers). The mobile client accesses to the Internet cloud services through wireless networks with limited bandwidth. In the cloud are clusters of commodity servers which are interconnected to each other through high-speed switches. In the following, we describe the terminologies which will be used through this paper.

End-users and Application Providers. End-users refer to the person who consumes the service through their mobile devices that the application provide. Application providers refer to the person/organization who develops, deploys and operates the applications.

Application Instances. Application instance means one execution of the application by one particular end-user. Multiple end-users can run the same application, but each user has its own application instance. Application instances of end-users are different in term of the places hosting the application in the system. For example, assuming an application consisting of three components, c1, c2 and c3, end-user A may run the components c1 and c2 locally, and offloads the component c3 to the cloud, while end-user B may run the component c2 locally and components c1 and c3 remotely. In this case, we say user A and user B have different application instances. Application instance of an end-user may change temporally in the execution place of its components as the mobile device’s load and wireless bandwidth varies.

2.3 Design Objectives

We have identified the following three requirements for the MCC system design.

High Performance. Two measures are commonly used to evaluate the performance of data stream applications, makespan and throughput. Makespan is the time used to process a single unit of data. It represents the responsiveness of the application. Throughput is the rate at which the input data is processed, and it determines the quality of result of the application. Taking an example of the gesture recognition application, the recognition accuracy will be better if the application can process more frames in one second. The MCC system is expected to provide maximal throughput for each end-user while satisfying the constraint of make-span.

Low Operational Cost. In a MCC system, the resources are provisioned at clouds to accommodate the computation offloaded from the end-users. The resource cost for operations is another critical factor we need to consider in the system design. Assuming that the application provider leases the cloud resources to host the application, the operational cost is measured by the amount of the leased cloud resources. It is required that the MCC system guarantees the minimal cost under given loads from end-users.

Adaptivity and Elasticity. The key benefit of the mobile cloud system is its combinational property of adaptivity and elasticity. Adaptivity means computations (or loads) that are offloaded from the end-user’s mobile device to the cloud are adaptive to the end-user’s changing mobile environment. For example, when the end-user’s device has a high CPU load and good networking bandwidth, most computations may be offloaded onto clouds; conversely, when the end-user’s device has an idle CPU and its networking bandwidth is low, most computation may be executed locally. Elasticity means the cloud resources can be provisioned cost-efficiently to meet the end-users’ offloading loads. The two properties make our proposed MCC system unique compared with existing system providing mobile services or cloud services.

To achieve the above goals we develop two key techniques, namely Adaptive Computation Partitioning and Multi-tenancy Component as a Service (CaaS). In the first technique, the device characteristics and networking bandwidth are profiled online on the user’s device. The partitioning algorithm is triggered on cloud as long as the variance of these profiling parameters exceed the threshold. The technique enables the mobile user to achieve an optimal partitioning whenever the user’s environment changes. The second techniques allow the application instances of different users to share the same component on cloud. Because of the component sharing, we do not need to separately allocate resources for each user to accommodate everyone’s peak data rate, but only need to allocate the resources such that it can serve the peak of the total data rate. Besides, this technique can avoid the frequent loading and unloading of some ‘hot’ components. Hot means those components are offloaded onto cloud by a majority of the mobile users.

3. ARCHITECTURAL DESIGN
The runtime framework consists of software modules on both the mobile side and the cloud side. The client side monitors the CPU workload and networking bandwidth. When the application is launched on the mobile client, an request is sent to the Resource Manager in cloud for augmented execution. The resource manager then assigns an Application Master to handle the request. The application master first asks the mobile client for its device characteristics such as CPU capability $p$, its workload $\eta$, and the current network bandwidth $B$. Using these dynamic information from mobile device as well as the static application properties stored in cloud, the application master then generates an optimal partitioning result by Algorithm 1, which is presented in Section V. The components assigned to the client are initiated as threads on the mobile device. Other components assigned to the cloud are invoked as services, namely Component-as-a-Service (CaaS). The application master is also in charge of the data transmission between the mobile client and cloud. In the framework, every mobile application has an Application Master in cloud to augment its execution. The components are shared and invoked by applications as a service in cloud. Resource Manager and the per-machine Node Manager, which monitors the processes on that machine, constitute the computation fabric in the cloud. Resource Manager manages the global assignment of computing resources to Application Masters and CaaS through cooperation with Node Managers.

### 3.1 Adaptivity of Partitioning

The application master, in the middle of the mobile clients and the cloud CaaSs, has two distinct functionalities: (a) to determine an optimal partition results and make the partitioning adaptive to the mobile client’s varying environment (local CPU load and wireless networking bandwidth); (b) to coordinate the distributed execution of the dataflow application.

Fig. 4 shows the software modules on both the mobile client and application master, which provides support for the adaptive partitioning. It is assumed that two logical communication connections exist between both sides: an "always-on" connection but low data rate wireless connection which is for transmitting the control message; another wireless connection with bandwidth $B$, which is to pipeline the data streams between the mobile client and cloud. The profiler on the mobile client measures the device’s characteristics at startup and continuously monitors its CPU workload and wireless network bandwidth. The controller on mobile client side maintains some thresholds on the variance of the profiling parameters. If any of the parameters increases/decreases by a value exceeding the threshold, a request for updating the partitioning result will be sent to the controller on the application master. The controller of application master calls optimization solver to generate a new partitioning result. The optimal partitioning algorithm will be described in Section V. Taking the result as the input, the underlying module DF Execution provide runtime support for the distributed execution of the dataflow application.

In the design of our framework, we make sure that the runtime software will not bring much burden onto the mobile device and should be as lightweight as possible. So we put the optimization solver on the cloud rather than the mobile device to reduce the local resource utilization. Although the design feature requires an always-on connectivity, it is reasonable because unless there is wireless connectivity, all the components of the dataflow application is executed locally by default without the need to call the optimization solver.

Besides, the partitioning results for different mobile environments are able to be backed up in the cloud storage. If the request for updating the partitioning has similar input parameters as previous ones, the partitioning result will be directly queried from the back up storage instead of being computed by the optimization solver. The back up mechanism reduces the latency of the partitioning.

### 3.2 Distributed Execution

Fig. 5 shows the distributed execution of dataflow example with two partitioning cases. In the framework, the local components run as threads on mobile device while the remote components are executed through the invocation of CaaS. In a partitioned dataflow application, we name the component allocated onto mobile device as local component, and the one offloaded onto cloud as remote component. The application master has one thread for every remote component. These threads are responsible for data transmission as well as CaaS invocation. Since the threads serve as the images of the remote components, we call them as image components.

In a partitioned dataflow graph, the shaded node represents the remote component; the blank one is the local component. The channels are classified into two categories, crossing channel and internal channel. The crossing channel, e.g., (2,3), (2,4), (7,8) in graph (a), refers to the edge in the graph which connects a local component and remote

**Figure 3:** Overview of the application framework

**Figure 4:** Cooperation between the mobile client and the application master
Component while the internal channel connects two local components, e.g., (1, 2) in graph (a), or two remote components, e.g., (4, 5) in graph (a). The crossing channels are implemented by TCP pipes. Through the TCP pipe, the data is pushed from one component to its successor. Each TCP pipe has one in-memory FIFO at the receiver side to buffer the data that may not be processed. The internal channels are implemented by shared memory FIFOs. As a result of the FIFOs on all the channels, our framework enables an asynchronous and loosely decoupled way to execute the concurrent components.

3.3 Multi-tenancy CaaS

We realize multi-tenancy feature for the CaaSs to allow multiple tenants or application instances to share the component. As shown in Fig.3, the multi-tenancy CaaSs implementation adopt a master and slave architecture, in which Component Slaves are real entities to do the computation and Component Master takes charge of scheduling tenants’ loads onto the component slaves. Specifically, Component Master negotiates resources from Resource Manager and work with Node Managers to launch/terminate component slaves according to the current request load. The purpose of the multi-tenancy CaaS is to guarantee an elastic utilization of underlying resources to accommodate the scalable CaaS requests.

In our framework, the end users have different application instances even when they run the same application. Each application instance consists of a set of components. The CaaS at the cloud side is usually shared by multiple application instances. According to the partitioning mechanism, application instances have various load requirements on one specific CaaS. The load requirements mean how fast the CaaS is required to process the input data stream.

In order to save the resources, we need to solve the load scheduling problem. The problem is to schedule various loads from the application instances onto the component slaves, such that the number of utilized component slaves are minimized. We assume that the component slaves have the same capacity. The load scheduling problem can be modeled as a Online Bin Packing Problem [21].

4. OPTIMAL PARTITIONING ALGORITHM

In this section, we describe the models, formulation and algorithm for solving the computation partitioning problem. The problem is formulated as an optimization problem, and the proposed algorithm is executed online by the Application Master shown in Fig.3.

Application Model. The application is modeled as a specific dataflow graph \( G = (V, E) \), where \( V \) has \( \{i \mid i = 1, 2, ..., n\} \) represents its components and \( E = \{(i, j) \mid i, j \in V\} \) represents the dependency between the components. \( s_i \) is the average number of CPU instructions required by component \( i \) to process one unit of data. \( d_{i,j} \) presents the amount of data required to be transmitted on the channel \((i, j)\) for one unit of data. The weight on a node \( i \), denoted as \( w_i \), represents the computational cost (time). The weight on an edge denoted as \( c_{i,j} \) is the communication cost (time). Both \( w_i \) and \( c_{i,j} \) are measured by one unit of data.

Throughput Model. Here, the throughput of the application is the objective for optimization. We define critical component/channel, which represents the component/channel that has the greatest weight among all the components/channels. Assuming that all the components’ capacity is unlimited and whatever level of pipeline parallelism is allowed, the throughput of the dataflow application is determined by the critical component/channel, which have the slowest speed to compute/transfer the data. So we have the formula for throughput \( TP = \frac{1}{\eta} \), where

\[
tp = \max\{\max_{i \in V} (w_i), \max_{(i,j) \in E} (c_{i,j})\}.
\]

Offloading Model. The offloading decision is made mainly depending on the local computing resources and the wireless networking quality. A few parameters are introduced to model these properties. \( p \) is the CPU’s capability of the mobile device, measured by the number of performed instructions per second. \( q \) is the percentage of the ideal CPU resource. It also indicates the current working load on the mobile device. So the available CPU resource on the mobile device is \( qp \). \( B \) is the bandwidth of the wireless network for the mobile device to access the Internet cloud. We have the following assumptions in our system model. i) The components running concurrently on the mobile devices are allocated equal CPU resources. ii) If a component is offloaded onto cloud, other components running on the mobile client will speed up because of the acquisition of the released CPU resources. The speedup factor is \( \frac{N}{N-1} \), where \( N \) is the number of components on the mobile device before the offloading event. iii) The cloud always has abundant resources to accommodate the offloaded components such that they will not become the critical component in the dataflow graph. iv) The total wireless bandwidth \( B \) are shared by all the crossing channels, where crossing channel in the dataflow graph is defined as the one which connects two components residing two sides of different resources. It is possible to allow for the mobile device to allocate disparate bandwidth to different crossing channels. We do not distinguish between the uplink and downlink bandwidth in our model. v) If interdependent components are offloaded onto cloud, the channels connecting between them in the cloud will not become the critical channel. vi) The input data of the application is acquired from the sensors on the mobile device, and output data should also be delivered to the mobile device.

Problem Formulation. Given the dataflow application
$\{G(V,E), s_i, d_{i,j}\}$, the mobile device properties $\{p, \eta\}$, and the wireless network bandwidth $B$, the partitioning problem in this study is the problem of allocating a set of $v$ components of the data flow graph to the resources (the mobile client and the cloud) and allocating the limited wireless bandwidth $B$ to the potential crossing channels such that the throughput of the data stream application is maximized. The optimization problem is formulated in Equation (2).

$$\max_{x_i,y_{i,j}} TP = \frac{1}{t_p}, \quad i,j \in \{0,1,\ldots,v+1\},$$  
$$t_p = \max_{i \in \mathbb{V}} \left\{ \sum_{(i,j) \in \mathcal{E}} y_{i,j} (x_i - x_j)^2 = B, \right.$$  
$$y_{i,j} > 0,$$  
$$s.t.: \quad x_0 = 1,$$  
$$x_{v+1} = 1,$$  
$$x_i = 0 \text{ or } 1, \quad i \in \{1,2,\ldots,v\}$$  

(2)

The core variables are $x_i$ and $y_{i,j}$. $x_i$ is either 0 or 1 integer, indicating the offloading decision for component $i$. If $x_i$ equals to 1, component $i$ is executed on the mobile device; otherwise $x_i = 0$ means running on the cloud. $y_{i,j}$ is the wireless bandwidth allocated to the channel $(i,j)$. Note that two virtual nodes, 0 and $v+1$, are created to satisfy the constraint that the input/output data of the application should be forwarded to the mobile device. Two edges (0,1) and (v, $v+1$) are added into the set of edges $\mathcal{E}$ of the data flow graph, where node 1 is the entry node and node $v$ is the exit node. Accordingly, $d_{0,1}$ is the size of an unit of input data, $d_{v+1}$ is the size of an unit of output data.

Algorithm Design. The objective function shown in Equation (2) depends on two variables, $x_i$ and $y_{i,j}$. We first study the problem of allocating the wireless bandwidth $B$ to the crossing edges given a specific partition. It is not difficult to prove theorem 1.

**Theorem 1**: Given a partition $X = \{x_i | i = 1,2,\ldots,v\}$, the throughput is maximized when $y_{i,j}$ satisfies the condition that

$$y_{i,j} = \frac{d_{i,j}}{t_{\text{comm}}(X)}, \quad \forall (i,j) \in \mathcal{E} \quad \text{and} \quad x_i \neq x_j,$$  
$$y_{0,1} = 0, \forall (i,j) \in \mathcal{E} \quad \text{and} \quad x_i = x_j,$$  
$$y_{v+1,i} = (1 - x_i) \frac{d_{i,v+1}}{t_{\text{comm}}(X)},$$  
$$y_{i,v+1} = (1 - x_i) \frac{d_{v+1,i}}{t_{\text{comm}}(X)};$$  

(3)

where $t_{\text{comm}}(X)$ is the communication cost/time that each crossing channel needs to transfer data of unit, $t_{\text{comm}}(X) = 1 \frac{1}{B}(1 - x_i)d_{0,1} + (1 - x_v)d_{v+1} + \sum_{(i,j) \in \mathcal{E}} (x_i - x_j)^2d_{i,j}$.  

(4)

So, the original problem can be reduced into

$$\max TP = \frac{1}{\max\{t_{\text{comm}}(X), t_{\text{comm}}(\bar{X})\}}$$  

(5)

where $t_{\text{comm}}(X) = \max_{x_i \in X} \left\{ \frac{1}{\eta p} \sum_{i=1}^{v} x_i \right\}$ and $X$ is a $v$-dimension vector of 0 and 1. The application throughput is constrained either by the speed that the local components process the data or by the speed the crossing channels transfer data.

The evolution starts from a randomly generated population (line 1). $NIND$ represents the number of the individuals of the population. In each generation, the fitness of every individual in the population is evaluated (line 4-5). Individuals are probabilistically selected from the current population for breeding according to their fitness (line 6). We use roulette wheel selection in our algorithm. The probability that each individual is selected is proportional to its fitness. Generation gap $GGAP$ is a control parameter of our algorithm, which represents the number of selected individual divided by the current population size. The selected individuals are modified through crossover (line 7-11) and mutation (line 12-14), and added into the current population. Our genetic algorithm evaluates the fitness of all individuals and selects the best ones with constant size of population (line 16). The best individuals serve as the new generation of partitions, which is then used in the next iteration of the algorithm. The algorithm terminates when the number of generations has reach certain upper bound $MAXGEN$. In the last generation, the partition with the highest throughput is chosen as the final partition (line 19-21).

Recall that in each round, our algorithm modifies current individuals using crossover and mutation. Crossover generates new individuals by combining two randomly selected
individuals (partitions), say A and B. During crossover, a randomly chosen gene position divides the binary string of A and B into two parts. One new individual obtains the first section of string from A and the second section of string from B. The second new individual obtains the inverse genes. Mutation takes the string of an individual and randomly changes one or multiple values. The mutation rate $\text{MUTR}$ is defined by the ratio of the amount of the changed bits to the total amount of bits in one chromosome.

5. NUMERICAL EVALUATION

We evaluate the proposed partition algorithm in this section. The performance metric we consider in the evaluation is the throughput of the data stream application.

5.1 Methodology

First, we evaluate how the controlling parameters of the algorithms, $\text{NIND}$ and $\text{GGAP}$, affect the performance. Second, we study the effect of the input parameters of our algorithms including application graphs, the wireless networking bandwidth $B$ and the available computing resource at mobile device $\eta_p$. At last, we demonstrate the factor that can affect the computational cost of our algorithm.

The input application graph we consider is the randomly generated application graphs. We have implemented a graph generator to generate the weighted streaming application graphs. We use the level-by-level method to create the graph which was proposed by Tobita and Kasahara[14]. We could control the graph that we want to generate through the following parameters: 1) number of nodes; 2) average out-degree; and 3) communication-to-computation ratio (CCR), where CCR is defined as the ratio of the average communication time to the average computation time as shown in equation (6).

$$\text{CCR} = \frac{\left[d_{0,1} + d_{v,v+1} + \sum_{(i,j)\in E} d_{i,j}\right]/[(e + 2) \times B]}{(\sum_{v\in V} \langle v \rangle)/B}$$

(6)

We have done a group of simulation to evaluate the effect of both controlling parameters and input parameters to the performance. Table I shows the configuration of our simulations. In each simulation, we choose one parameter as the variable, which is indicated by ‘*’, while assigning other parameters as constant values. For example, in No. 5 experiment, we study the effect of wireless bandwidth B onto the performance. In our configuration, we treat $\eta_p$ as one single parameter which indicates the available CPU resources on mobile device. Note that $B$ and $\eta_p$ shown in the table is a normalized value.

5.2 Results

We first presents the result of how the controlling parameters, $\text{NIND}$ and $\text{GGAP}$, affect both the throughput and the number of iterations that the genetic algorithm needs to converge. Fig.6(a) shows the throughput value in each iteration of our algorithm for different $\text{NIND}$s. The configuration of other parameters is shown in Table I under row No.1. We can see that larger $\text{NIND}$ value leads to better result. The algorithm takes fewer iterations to converge to the final throughput in case of higher $\text{NIND}$ value. Fig.6(b) shows the effect of $\text{GGAP}$ on the performance. Larger $\text{GGAP}$ value has both better convergence speed and the final throughput. It indicates that if more individuals are selected from the population for breeding in each iteration, the genetic algorithm will take fewer iterations to find the optimal individual.

We then present the effect of the input parameters on the performance. It contains the application graph properties (graph size $v$ and CCR), $\eta_p$ and $B$. The results are compared with other two intuitive strategies with no partitioning: 1) running all the nodes of the application graph on the cloud; 2) running all locally on the mobile device. At first, we study the relationship between the throughput and application graph size $v$. The configuration parameters are shown in Table I under No.3 Row. We increase $v$ while keeping the average node’s computational cost and communication cost of the edges not variable. So $v$ actually indicates the overall computational complexity of the application. Fig.6(c) shows that our algorithms can typically achieve more than 2X better throughput than other two strategies. It also shows that given the resources $B$ and $\eta_p$, the application performance of the partitioning scheme goes down as the application size rises up. When the application size becomes very large, our method tends to have the same performance with the all-cloud method. It is because when the application becomes extremely computational complex, offloading all the nodes of the application graph onto cloud can save much more computational cost than the overhead of communication, while partitioning method has little improvement on the performance in this case.

Fig.6(d) and Fig.6(e) respectively shows the influence of networking resource $B$ and computing resources of the mobile device $\eta_p$. It is interesting to find that our method always achieve better performance as $B$ increases while on the other hand, increasing $\eta_p$ does not necessarily lead to better performance. The different results can be explained as follows. For an optimal partition, the total throughput of the application is either limited by the computation or the communication as explained in Equation (5). In the former case we say the bottleneck is at computation while in the latter case we say the bottleneck is at communication. If the bottleneck is at communication, increasing bandwidth $B$ is definitely able to improve the performance; otherwise, we could always reduce the computation overhead $t_{comp}$ by moving one node from mobile to cloud. Normally this moving operation is likely to increase the communication overhead $t_{comm}$, but we have the increasing $B$ to accommodate the extra communication overhead. That’s why the increase of $B$ usually leads to the raising of the overall throughput. However, it is not guaranteed to reduce the communication overhead $t_{comm}$ by moving one node from cloud to mobile when the bottleneck is at communication. So in this case
the increase of np can not improve the overall throughput. Fig.6(c) indicates only in the case where np is large enough, the throughput of our method sensitively increases as the np increases, because in this case the result of our method approximates the 'all-mobile' method.

Fig.6(f) shows how CCR affects the performance. We obtain different CCRs in our simulation by changing $s_i/d_{ij}$ while keeping $B$ and np constant. It shows as the CCR rises the performance of our method first goes down and then rebounds. When CCR is large, running all nodes on the mobile side approaches the optimal performance. Fig.7(g) presents the number of nodes allocated onto mobiles device by our method in case of different CCRs. Obviously more nodes are executed on the mobile device when the CCR increases.

At last, we study the computational cost of our algorithms. Given the internal parameters such as NIND and GGAP, the computation cost is measured by the number of generations that our algorithm demands to converge. In practice it is not necessary, if not impossible, to cost a lot of computation to achieve the theoretical optimal throughput. We usually take a critical point, for example 90 percentage of the optimal value, as the actual convergence point. Fig.6(h) shows the generations required to achieve the convergence point for different application graph size v. Our algorithms have larger computational cost as the application graph size increases.

6. EXPERIMENTAL EVALUATION

We conduct a series of experimental tests on real-world applications to validate the results. We take one simple application as our test examples: QR code recognition. The application is re-written using the Flow-based Programming (FBP) model [8], in which application is modeled as a set of functional components running in parallel and a set of channels streaming data from one component to another.

6.1 QR-code Recognition

We spend more than one month to program the QR-code recognition application with the FBP model. The application consists of three phases: image capturing, image pre-processing, and QR code decoding. In the program, we decompose the three phases into 9 functional components, which are shown in Fig.7. For convenience of description, we label each component with a circled number. Ⓚ and Ⓣ are respectively the entry and exit component of the program. We measure the size of data that are transferred between the components with a $640 \times 480$ (300K Bytes) input image. The results are shown by the labels on the edges in Fig.7. Note that the size of data transferred between components depends on the size of input image. In our experiment, all the tests use the 300K Bytes input image.

6.2 Experiment setup and Results

In the experiment environment, we use the Motorola MB510
Table 2: Local computational time of components

<table>
<thead>
<tr>
<th>Component No.</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>130</td>
</tr>
<tr>
<td>3</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>280</td>
</tr>
<tr>
<td>8</td>
<td>427</td>
</tr>
<tr>
<td>9</td>
<td>427</td>
</tr>
<tr>
<td>10</td>
<td>240K</td>
</tr>
<tr>
<td>11</td>
<td>40K</td>
</tr>
<tr>
<td>12</td>
<td>40K</td>
</tr>
<tr>
<td>13</td>
<td>1.2M</td>
</tr>
<tr>
<td>14</td>
<td>1.2M</td>
</tr>
</tbody>
</table>

Table 3: Transmission time between the components

<table>
<thead>
<tr>
<th>Edges</th>
<th>Trans. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1→2</td>
<td>10240 ms</td>
</tr>
<tr>
<td>1→3</td>
<td>12800 ms</td>
</tr>
<tr>
<td>2→3</td>
<td>427 ms</td>
</tr>
<tr>
<td>3→4</td>
<td>427 ms</td>
</tr>
<tr>
<td>3→5</td>
<td>427 ms</td>
</tr>
<tr>
<td>4→7</td>
<td>2 ms</td>
</tr>
<tr>
<td>5→7</td>
<td>2 ms</td>
</tr>
<tr>
<td>7→8</td>
<td>1 ms</td>
</tr>
</tbody>
</table>

Table 4: Partitions under different bandwidths

<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 Kbps</td>
<td>Mobile:{1, 2, 3, 4, 5, 6}, Cloud:{7}</td>
</tr>
<tr>
<td>240 Kbps</td>
<td>Mobile:{1, 2}, Cloud:{3, 4, 5, 6, 7}</td>
</tr>
<tr>
<td>1.2 Mbps</td>
<td>Mobile:{1}, Cloud:{2, 3, 4, 5, 6, 7}</td>
</tr>
</tbody>
</table>

Android phone as the mobile device. The cloud resources contain a cluster of PCs in our lab. The wireless connections between mobile and cloud are through WLAN or 3G. The open source runtimes JavaFBP [22] are deployed on both the Android phone and cloud nodes to support the execution of the FBP programs. Java Message Service (JMS) [23], a Message-oriented Middleware, is also installed together with JavaFBP on both the mobile and cloud nodes to take charge of the data transmission between the distributed components.

First, we profile the computational cost of each component on the mobile device. We run the QR-code recognition program on the Android phone for 30 times. Table 2 shows the average time that each component needs to process one 300K Bytes image. Then, we measure the communication cost between the components. It is equal to the data size between two components divided by the wireless bandwidths. Table II shows the data transmission time between two dependent components under the bandwidth 240 Kbps.

Now we start the system, and demonstrate how the partition changes as the wireless bandwidth varies. Table 4 records the partitions under various wireless bandwidths. We can see that when the network bandwidth is as low as 40Kbps, all the components except 7 are executed on the Android phone. As the bandwidth increases, the optimal partition includes more components running on the cloud side.

We also compare the performance of the partitioned application with other two strategies without partitioning. From the results shown in Fig.8, we conclude that for the QR code recognition application, the partitioned execution can achieve at least 2X better throughput in reality than the executions without partitioning. For example, when the bandwidth is 240Kbps, the application with optimal partition can process 2.4 images per second, while the application throughput is 0.5 images per second if all the components are executed on the mobile device, and approximates to 0.1 if all the components are executed on the cloud.

Figure 8: QR-Code Recognition Performance

7. RELATED WORKS

This paper is most related to computation partitioning in mobile computing. Other related works include the large scale cloud application frameworks. We introduce the related works on these topics.

Computation offloading is the most widely used technique to solve the resource poverty problem of mobile devices in mobile cloud computing environment [17][18]. Karthik et. al [19] argues that cloud computing could potentially save energy for mobile user, but not all applications are energy efficient when migrated to the cloud. It depends on whether the computation saved due to offloading outperforms the communication cost. M. Satyanarayanan [20] presents a computing model that enables a mobile user to exploit Vms to rapidly instantiate customized service software on a nearby cloudlet and uses the service over WLAN. Rather than relying on a distant cloud, the cloudlets avoid the long latency introduced by wide-area networks for accessing the cloud.

Compared with offloading a whole application into cloud, a partitioning scheme is able to achieve a fine granularity for computation offloading. [16] are the early works to study the application partitioning problem. These works only consider the static partitioning problem. [4][1][15][6] extends these work to support dynamic application partitioning. They argue that mobile clients could face wide variations and rapid changes in network conditions and local resource availability when accessing remote data and services. In order to enable applications and systems to continue to operate in such dynamic environments, the computation of clients and cloud has to be adaptive in response to the changes in mobile environments. However, the works [4][1][15][6] advocate the used partitioning scheme that assumes the computation components are sequentially executed among the mobile device and cloud. Since they do not take advantage of the parallelism choices, these methods are not suitable for mobile data stream applications.

The most similar work to ours is Odessa [12] which provides a runtime system that is able to adaptively make offloading and parallelism decisions for mobile interactive perception applications. Their main concern is on the makespan of the application rather than the throughput. Moreover, it is not guaranteed that the proposed incremental greedy algorithm can always converge to a optimal partitioning result with limited overhead. The advantage of our work is that our framework allows the partitioning algorithm to run on the cloud side. The powerful computing resource at cloud side guarantees that the genetic algorithm is more likely to
converge to the global optimal partition. Furthermore, all
the systems in [4][11][15][6][12] are designed based on the tra-
tditional mobile computing paradigm and lack the solution
for scalability.

A few related works focus on the design of application
frameworks in cloud computing. The most popular one
is MapReduce [7] from Google. It provides a simple pro-
gramming abstraction and hides many messy details of dis-
tributed and parallel computing from the developers. Dryad
[11] allows a more general application model than MapRe-
duce. The job fitting in this framework consists of an acyclic
dataflow graph of sequential processing modules. Both MapRe-
duce and Dryad are suitable for offline analysis of large data
sets (batch computation system) rather than interactive ap-
lications (real-time operation/query system). These mod-
els are not well suitable for developing mobile streaming ap-
plications. Systems such as Aurora [10], Borealis [2], and
TelegraphCQ [13] provide support for continuous queries
over data streams. The systems process streaming data, per-
form runtime adaptation, and consider real-time constraints.
However, these works do not fit well in the MCC because it
assumes all the computation is performed in cloud side and
do not provides a flexible and adaptive mechanism to run
the computation across the client and clouds.

8. CONCLUSIONS

In this paper we study the computation partitioning prob-
lem for mobile data stream applications. We have designed
a cloud-based framework to provide runtime support for the
adaptive partitioning and distributed execution of such ad-
vanced mobile cloud applications. The framework is able to
serve large number of mobile users by leveraging the elas-
tic resources in existing cloud infrastructures. Under this
framework, we also have designed a genetic algorithm to
solve the partition problem. Both numerical evaluation and
real world experiment results show that our method can
provide more than 2X improvement in the application per-
formance over the methods without partitioning.

9. ACKNOWLEDGMENTS

This work is financially supported by Hong Kong RGC
under GRF Grant No. 510611E, and Microsoft under Grant
No. H-ZD89.

10. REFERENCES

[22] Java FBP: http://www.jpaulmorrison.com/cgi-
bin/wiki.pl?JavaFBP
http://www.oracle.com/technetwork/java/jms/index.html