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

Lei Yang*, Jiannong Cao*, Shaojie Tang†, Tao Li*, Alvin T. S. Chan*
*Department of Computing, The Hong Kong Polytechnic University, Hong Kong
†Department of Computer Science, Illinois Institute of Technology, Chicago, USA
*{csleiyang, csjcao, cstaoli, cstschan}@comp.polyu.edu.hk, †tangshaojie@gmail.com

***Abstract*—The advances in technologies of cloud computing and mobile computing enable the newly emerging mobile cloud computing paradigm. Three 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 the computation partitioning, which aims at optimizing the partition of a data stream application between mobile and cloud such that the application has 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 in other applications. We first propose a framework to provide runtime support for the dynamic 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 to perform the optimal partition. We have conducted extensive simulations. The results show that our method can achieve more than 2X better performance over the execution without partitioning.**

***Keywords*-mobile cloud computing; application partitioning; genetic algorithm;**

I. Introduction

Cloud computing is an important transition and 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] [5]; 3) augmenting the execution of mobile applications using cloud resources, e.g. by offloading selected computing tasks required by applications 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 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 requirements. 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 interme-

diate object poses or human gestures. Second, the computer vision and machine learning algorithms used to process this streaming data are computation-intensive.

However, along with the great benefit brought by cloud, there is significant complexity involved in ensuring that mobile applications and associated data perform and adjust when needed to achieve efficient operation under variable loads. Specifically, there are two challenges to address. First, on the mobile side, we need to handle the wide variations and dynamic changes in network conditions and local resource availability. In order to achieve high performance, the decision of which units of computation should be moved to the cloud has to be made adaptive to the changes in mobile environments. (We refer it as *application partitioning* between mobile client and cloud.) Second, for the cloud side, we need to handle the unpredictable and varying load from multiple mobile clients of the application.

So far, there is no comprehensive treatment of the fore-mentioned problems. Existing computation partitioning mechanisms enable an adaptive execution of mobile application between mobile devices and the server [4] [1] [15] [6] [12]. However, these efforts are only suitable in traditional mobile Internet computing and do not give any solution on how to use the elastic resources 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 provide a flexible and adaptive mechanism to schedule the computation across the client and clouds.

In this paper, we make use of the third MCC approach to design an execution framework for mobile data stream applications. The framework consists of runtime systems to support the adaptive partitioning and distributed execution of the applications. In order to achieve an efficient way to serve a large number of users, the framework adopts a multi-tenancy policy such that the computation instances on the cloud are able to be shared by multiple applications/tenants. Our contributions in this paper are as follows:

- We define the partitioning problem for mobile data stream applications. To the best of our knowledge, it is the first work to study the partitioning problem for mobile data stream applications which require parallel execution of different operations onto the streaming data to achieve high processing speed.
- We design a framework to facilitate the partitioning and execution of mobile data stream applications. Our framework do not only support the adaptive partitioning for a single user, but also supports efficient utilization of cloud resources to serve a scaling number of mobile users.
- We design a genetic algorithm to perform the optimal partition based on the application framework. Extensive simulations have been conducted to demonstrate the effectiveness of our algorithm.

In what follows, we first present the related works in Section II. We then describe the models and formulate the partitioning problem in Section III. The design of the execution framework and the algorithm is respectively described in Section IV and Section V. The evaluation results are presented in Section VI. We conclude the paper in Section VII.

II. 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 [18] [19]. Karthik et. al [20] 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. According to the cost model in this paper, the offloading will bring benefit to energy saving if the application has a large computation-to-communication ratio and runs in a networking environment with good connectivity. M. Satyanarayanan [21] 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. [17] [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] [1] [15] [6] [12] are designed based on the traditional 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 programming abstraction and hides many messy details of distributed and parallel computing from the developers. Dryad [11] allows a more general application model than MapReduce. The job fitting in this framework consists of an acyclic dataflow graph of sequential processing modules. Both MapReduce and Dryad are suitable for offline analysis of large data sets (batch computation system) rather than interactive applications (real-time operation/query system). These models are not well suitable for developing mobile streaming applications. Systems such as Aurora [10], Borealis [2], and TelegraphCQ [13] provide support for continuous queries over data streams. The systems process streaming data, perform 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.

III. Models and Problem Formulation

A. Application Model

The data stream application, presented as a directed acyclic dataflow graph $G = (V, E)$, is composed of a set of v *components* $V = \{i | i = 1, 2, ...v\}$ and a set of *channels* $E = \{(i, j) | i, j \in V\}$. (*Component* and *node* terms are interchangeably used.) 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.

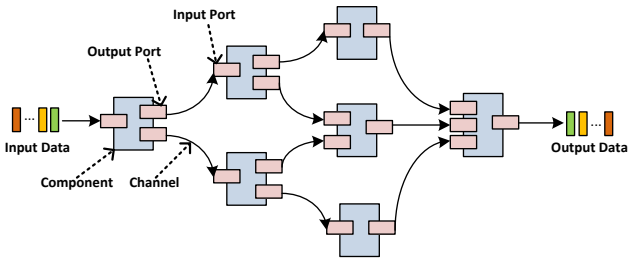


Figure 1. Application Model

The dataflow model is based on a data centric approach and usually takes advantage of pipeline to accelerate data processing.

Given a specific dataflow application, 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 presents 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.

Definition 1: *Critical path* is defined as the longest path from the entry node to the exit node, where the weight of a path is defined as the summation of the weight of all the nodes and edges on the path.

Definition 2: *Critical component/channel* in a dataflow graph is defined as the one which has the greatest weight among all the components/channels.

There are two metrics to evaluate the performance of the dataflow applications, *makespan* and *throughput*. *Makespan* presents the total time to process one unit of input data. It is equal to the weight of the critical path in the dataflow graph. *Throughput* presents the number of units of input data the dataflow is able to process per second. Assuming that all the channels' 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 $TP = \frac{1}{t_p}$, where

$$t_p = max\{\max_{i \in V}(w_i), \max_{(i,j) \in E}(c_{i,j})\}. \quad (1)$$

B. System Model

The mobile cloud system shown in Fig. 2, 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. Possibly the mobile client can accelerate the execution of the mobile applications by offloading computing tasks onto the cloud. The benefit of offloading is the computational time saved due to the faster execution on

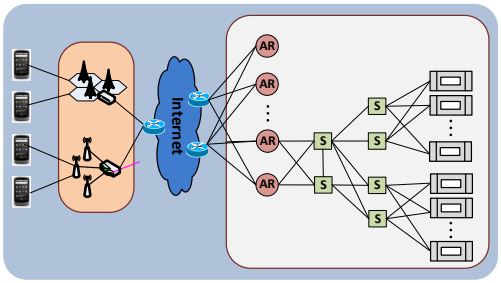


Figure 2. Mobile Cloud System Model

much more powerful CPUs in the cloud. The cost comes from data transmissions over wireless networks between the client and cloud. The question here is which components in the dataflow graph should be offloaded onto cloud such that the throughput of the application is maximized. The reason why we take the throughput as the objective is that the accuracy of many mobile data stream applications such as face/gesture recognition is determined by its throughput.

The offloading decision mainly depends 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 instructions per second. η 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 ηp. 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.

1) The components running concurrently on the mobile devices are allocated equal CPU resources.

2) 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.

3) The cloud always has abundant resources to accommodate the offloaded components such that they will not become the critical component in the dataflow graph.

4) 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 allowed 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.

5) If interdependent components are offloaded onto cloud, the channels connecting between them in the cloud will not become the critical channel.

6) 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.

C. 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 dataflow 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).

$$\max_{x_i, y_{i,j}} TP = \frac{1}{t_p}, i, j \in \{0, 1, \cdots, v+1\}, \text{ where}$$

$$t_p = \max\{\max_{i \in V}(x_i \cdot \frac{s_i}{\eta p} \sum_{i \in V} x_i), \max_{(i,j) \in E}(\frac{d_{i,j}(x_i - x_j)^2}{y_{i,j}})\},$$

$$s.t. \begin{cases} \sum_{(i,j) \in E} y_{i,j}(x_i - x_j)^2 = B, \\ y_{i,j} > 0, \\ x_0 = 1, \\ x_{v+1} = 1, \\ x_i = 0 \text{ or } 1, i \in \{1, 2, \cdots, v\} \end{cases} \tag{2}$$

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 from/delivered to the mobile device. Two edges $(0, 1)$ and $(v, v+1)$ are added into the set of edges E of the dataflow 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. $d_{v,v+1}$ is the size of an unit of output data.

IV. Execution Framework Design

A. Overview

Fig. 3 shows the overview of a dataflow execution framework in mobile cloud computing. 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 η, 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

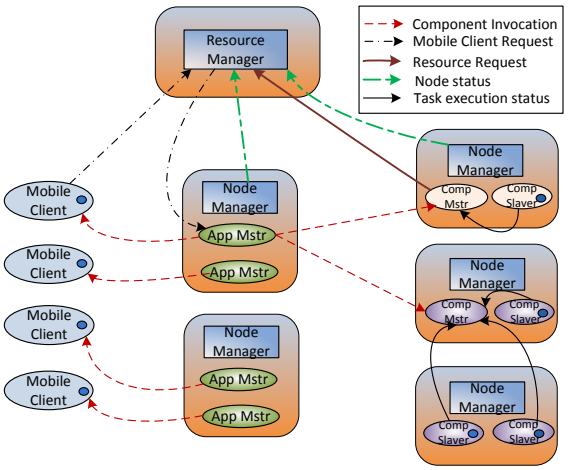


Figure 3. Overview of the application framework

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 CaaSs through cooperation with *Node Managers*. In our design, we realize multi-tenancy feature for the CaaSs, which allows multiple tenants/applications to share the CaaS instance. The instance is able to be replicated to handle the scaling-up load from tenants. A master and slave architecture is used to implement the multi-tenancy CaaSs, in which *Component Master* is responsible for scheduling and replicating the component instances (also referred to *Component Slaves*). Specifically, *Component Master* negotiates resources from Resource Manager and work with Node Managers to launch/terminate 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.

B. Adaptive 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

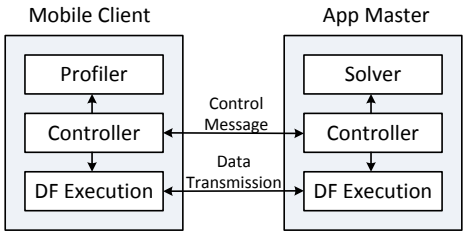


Figure 4. Cooperation between the mobile client and the application master

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. 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.

C. Distributed Execution

Fig.5 shows the distributed execution of dataflow example withtwo partitioning cases. In the framework, the local components run as threads on mobile device while the remote components are executed through the invocation of CaaSs. 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,

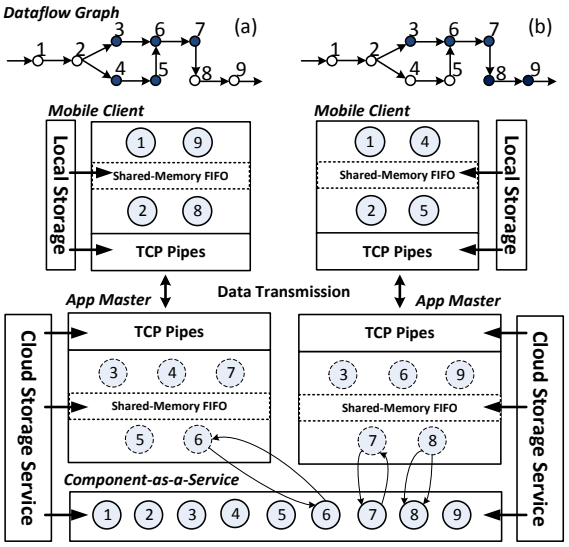


Figure 5. Distributed Dataflow Execution

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 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.

V. 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 $X = \{x_i | i = 1, 2, \cdots, v\}$, the throughput is maximized when $y_{i,j}$ satisfies the condition that

$$\begin{cases} y_{i,j} = \frac{d_{i,j}}{t_{comm}(X)}, \forall (i,j) \in E \text{ and } x_i \neq x_j, \\ y_{i,j} = 0, \forall (i,j) \in E \text{ and } x_i = x_j, \\ y_{0,1} = (1 - x_1)\frac{d_{0,1}}{t_{comm}(X)}, \\ y_{v,v+1} = (1 - x_v)\frac{d_{v,v+1}}{t_{comm}(X)}, \end{cases} \tag{3}$$

where $t_{comm}(X)$ is the communication cost/time that each crossing channel needs to transfer a unit of data,

$$t_{comm}(X) = \frac{1}{B}[(1 - x_1)d_{0,1} + (1 - x_v)d_{v,v+1} + \sum_{(i,j) \in E} (x_i - x_j)^2 d_{i,j}]. \tag{4}$$

So, the original problem can be reduced into

$$\max_X TP = \frac{1}{\max\{t_{comp}(X), t_{comm}(X)\}}, \tag{5}$$

where $t_{comp}(X) = \max_{x_i \in X}\{x_i \frac{s_i}{\eta p} \sum_{i=1}^{v} x_i\}$ and X is an 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.

```
   Input  : NIND, GGAP, MUTR, MAXGEN
   Output: The optimal partition
1  Chrom ← RandomlyCreatePopulation(NIND);
2  Gen ← 0;
3  while Gen < MAXGEN do
4      ObjV ← GetThroughputOf(Chrom);
5      FitnV ← Normalize(ObjV);
6      SelCh ← RouletteWheelSelect(Chrom, FitnV, GGAP);
7      i ← 1;
8      while i < GGAP * NIND do
9          CrossingOver(SelCh[i], SelCh[i + 1]);
10         i ← i + 2;
11     end
12     for i ← 1 to GGAP * NIND do
13         SelCh[i] ← Mutation(SelCh[i], MUTR);
14     end
15     ObjVSel ← GetThroughputOf(SelCh);
16     Chrom ← Reinsert(Chrom, SelCh, ObjV, ObjVSel);
17     Gen ← Gen + 1;
18 end
19 ObjV ← GetThroughputOf(Chrom);
20 i ← GetIndexOfMaximum(ObjV);
21 return Chrom[i];
```

Algorithm 1: The genetic algorithm for partitioning

We then propose a genetic algorithm to solve the reduced partition problem as shown in Algorithm 1. Essentially, we treat different partitions as a population of individuals with different chromosomes. Individuals with higher fitness as defined by Equation (5) in this work are more likely to survive and multiply. A partition X is represented by binary string $X = \{x_1, x_2, \cdots, x_v\}$. The encoding method successfully identifies the allocation of either on the mobile device side or otherwise. For example, if x_i equals to 1, component i is executed on the mobile device; otherwise it should be put in the cloud.

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 generic 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 in 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 $MUTR$ is defined by the ratio of the amount of the changed bits to the total amount of bits in one chromosome.

VI. Evaluation

A. Metric and Methodology

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. First, we evaluate how the controlling parameters of the algorithms, $NIND$ and $GGAP$, 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 ηp. 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). If an application graph's CCR is high, it can be considered as a communication-intensive application; otherwise, it is an computation-intensive application.

Table I
Configuration in each experiment

No.	NIND	GGAP	MUTR	G(v, d_{out}, CCR)	B	ηp
1	30	*	0.02	G(30, 3, 1)	1	1
2	*	0.8	0.02	G(30, 3, 1)	1	1
3	80	0.9	0.02	G(*, 3, 1)	1	1
4	80	0.9	0.02	G(50, 3, *)	1	1
5	80	0.9	0.02	G(50, 3, 1)	*	1
6	80	0.9	0.02	G(50, 3, 1)	1	*

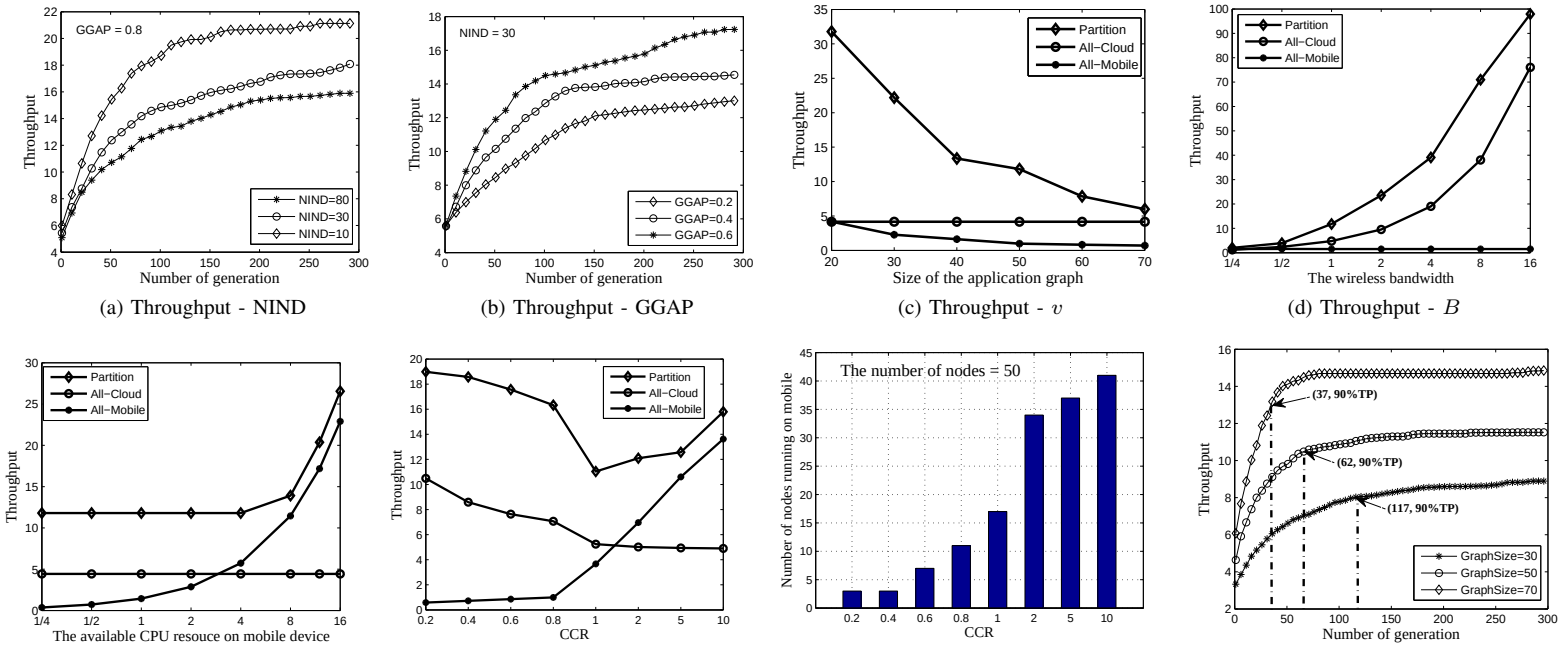
$$CCR = \frac{[d_{0,1} + d_{v,v+1} + \sum_{(i,j)\in E} d_{i,j}]/[(e+2)\times B]}{(\sum_{i\in V} s_i)/(v \times \eta p)} \quad (6)$$

We have done a group of experiments to evaluate the effect of both controlling parameters and input parameters to the performance. Table I shows the configuration of our experiments. In each experiment, 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 ηp as one single parameter which indicates the available CPU resources on mobile device. Note that B and ηp shown in the table is a normalized value.

B. Evaluation Results

We first presents the result of how the controlling parameters, $NIND$ and $GGAP$, 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 $NIND$s. The configuration of other parameters is shown in Table I under row No.1. We can see that larger $NIND$ value leads to better result. The algorithm takes fewer iterations to converge to the final throughput in case of higher $NIND$ value. Fig.6(b) shows the effect of $GGAP$ on the performance. Larger 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), ηp 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



(a) Throughput - NIND (b) Throughput - GGAP (c) Throughput - v (d) Throughput - B

(e) Throughput - ηp (f) Throughput - CCR (g) The number of nodes on mobile (h) Computational cost with different v

Figure 6. Simulation results

that our algorithms can typically achieve more than 2X better throughput than other two strategies. It also shows that given the resources B and ηp, 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 ηp. It is interesting to find that our method always achieve better performance as B increases while on the other hand, increasing ηp 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 ηp can not improve the overall throughput. Fig.6(e) indicates only in the case where ηp is large enough, the throughput of our method sensitively increases as the ηp 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 ηp as 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.

VII. Conclusions

In this paper we study the application partitioning problem for mobile data stream applications. We have designed an application framework to provide runtime support for the adaptive partitioning and distributed execution of such advanced mobile cloud applications. The framework is able to serve large number of mobile users by leveraging the elastic resources in existing cloud infrastructures. Under this framework, we also have designed a genetic algorithm to solve the partition problem. The simulation results show that our method can provide more than 2X improvement in the application performance over the methods without partitioning.

In our future work, we are at first going to conduct a series of experimental tests on real-world applications such as hand gesture recognition and mobile augmented reality and so on based using our proposed framework. In the experiments, the applications are written according the Flow-based Programming model [8], which models the applications as a set of fine-granularity functional components running in parallel and a set of channels streaming data from one component to another. We are testing the performance of the applications in case of various environment parameters such as mobile devices' CPU resource and networking bandwidth, and compare the performance of the partitioned application with other two strategies without partitioning. Another important objective in this work is to save the cost of cloud resources, which is quite important from the standpoint of application providers. Loading balancing and scheduling problem in the multi-tenancy CaaSs will be studied in our future work to minimize the application provider's operational cost while satisfying the maximum performance of the partitioned application.

Acknowledgment

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

References

[1] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, Ashwin Patti. CloneCloud: Elastic Execution between Mobile Device and Cloud. In *Proc. EuroSys'10*.

[2] D. J. Abadi, Y. Ahmad, M. Balazinska.etc. The Design of the Borealis Stream Processing Engine. In *Proc.Innovative Data Systems Research*, 2005.

[3] E. E. Marinelli. Hyrax: Cloud Computing on Mobile Devices using MapReduce. In *Master Thesis, Carnegie Mellon Universtiy*, 2009.

[4] Eduardo Cuervoy, Aruna Balasubramanianz, Dae-ki Cho. MAUI: Making Smartphones Last Longer with Code Offload. In *Proc. MobiSys'10*. ACM press, 2010.

[5] Gonzalo Huerta-Canepa, Dongman Lee. A Virtual Cloud Computing Provider for Mobile Devices. In *Proc. ACM MCS'10*, pages 3756–3761. ACM Press, 2010.

[6] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso. Calling the Cloud: Enabling Mobile Phones as Interfaces to Cloud Applications. In *Proceedings of Middleware'09*, pages 1–20. Springer, 2009.

[7] J. Dean, S. Ghemawat. MapReduce: simplified data processing on large clusters. In *ACM OSDI'04*.

[8] J. Paul Morrison. Flow-Based Programming: A New Approach to Application Development, 2nd Edition. In . CreateSpace, 2010.

[9] J. Rellermeyer, O. Riva, and G. Alonso. AlfredO: An Architecture for Flexible Interaction with Electronic Devices. In *Middleware'08*.

[10] M. Cherniack, H. Balakrishnan, M. Balazinska etc. Scalable Distributed Stream Processing. In *Proceedings of the Conference on Innovative Data Systems Research*, pages 422–440. ACM Press, 2003.

[11] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Prof. EuroSys'07*.

[12] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, Ramesh Govindan. Odessa: enabling interactive perception applications on mobile devices. In *ACM MobiSys'11*.

[13] S. Chandrasekaran, O. Cooper, A. Deshpande etc. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proceedings of the Conference on Innovative Data Systems Research*, 2003.

[14] T. Tobita, H. Kasahara. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling*, 5(5):379–394, 2002.

[15] Xinwen Zhang, Anugeetha Kunjithapatham, Sangoh Jeong, Simon Gibbs. Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mobile Networks and Applications*, 16(3), 2009.

[16] Zhiyuan Li, Cheng Wang, Rong Xu. Task Allocation for Distributed Multimedia Processing on Wirelessly Networked Handheld Devices. In *IPDPS'02*.

[17] Zhiyuan Li, Cheng Wang, Rong Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 1–6. IEEE Press, 2001.

[18] K. Yang, S. Ou, H.H. Chen. On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. *IEEE Communication Magazine*, 46(1):56–63, 2008.

[19] R. Wolski et al. Using Bandwidth Data to Make Computation Offloading Decisions. In *IPDPS'08*.

[20] Karthik Kumar, Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy. In *IEEE Computer Society*. IEEE Press, 2010.

[21] M. Satyanarayanan, P. Bahl, R. C?aceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 6(4):14–23, 2009.