Dynamic Computation of the Application Between Smart Mobile Device and Cloud Computing

R. Lakshman Naik\textsuperscript{a}, Dr. B. Manjula\textsuperscript{a}, M. Rajendra Prasad\textsuperscript{b}

\textsuperscript{a}Department of Computer Science, Kakatiya University, Warangal, A.P., India
\textsuperscript{b}Department of Computer Science, Alluri Institute of Management Sciences, Warangal, A.P., India

ABSTRACT

The hardware-software partitioning problem is a key aspect of co-design of digital electronic systems; extensive research has been performed with diverse definitions of partitioning problems. However, existent partitioning solutions are not applicable to many real-time applications partly because of restricted input specification or insufficient constraints. By using the off-loading technique, a fundamental problem is to partition the dynamic computation of application involved in between the mobile device and cloud.

In this paper, we proposed three approaches for mobile cloud applications: Extending the access to cloud services to mobile devices, to enabling mobile devices to work collaboratively as cloud resource providers, to enhance the execution of mobile applications on portable devices using available cloud resources. Suitable framework is providing for runtime support with dynamic computation of the application was proposed. This is different from existing mechanism, the framework is not only allows the dynamic partitioning for a single user but also supports sharing and computation instances among multiple users in the cloud to achieve efficient utilization of the underlying cloud resources.

Keywords: Mobile Cloud Computing, Application Processing and Offloading, FIFO, CaaS.

I. INTRODUCTION

Over the past few years, advances in the field of network based computing and applications on demand have led to an explosive growth of application models such as cloud computing, software as a service, community network, web store, and so on. As a major application model in the era of the Internet,
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 everyday 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: a) extending the access to cloud services to mobile devices; b) enabling mobile devices to work collaboratively as cloud resource providers [1], [2]; c) 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.

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 [3], [4], [5], [6]. 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 [7] [8] [9] [10] 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-tendency policy such that the computation instances on the cloud are able to be shared by multiple applications/tenants.

II. FRAMEWORK OF DYNAMIC COMPUTATION

A. Framework of Application

The data flow application, presented as a directed acyclic dataflow graph \( D_b = (C_n, C) \), is composed of a set of \( n \) components \( C_n = \{ i \mid i = 1, 2, \ldots, n \} \) and a set of channels \( C = \{ (i, j) \mid i, j \in C_n \} \). (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.

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 \( w_{e_{i,j}} \) is the communication cost (time). Both \( w_i \) and \( w_{e_{i,j}} \) are measured by one unit of data.

- 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.
- 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 has the slowest speed to compute/transfer the data. So we have the formula for throughput

\[
T_p = \frac{1}{t_p} \quad \text{where} \quad t_p = \max_i \max_{c} \left( w_i \right) \quad \max_{i,j} \left( w_{e_{i,j}} \right)
\]

\[
\text{........... (1)}
\]

Fig.1: framework of application          Fig. 2: framework of Mobile Cloud System

B. Framework of System

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 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. \( \delta \) 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 \( p \delta \). \( B \) 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 of Partitioning

We define an application as a collection of processing modules interacting with each other. For example, modules are functions, and when a function calls another function, the two functions interact. For a typical client-server application, both the client and server modules belong to the application. The client runs at the weak device, and the server runs in machines in the cloud. We define a partition between client and server as the two sets of modules that cover all modules of the application. Formally, we define \( D \) as the entire set of modules, define \( d \) as the partition for mobile device, and define \( s \) as the partition for the server running in the cloud; \( d \cup s = D \). Fig. 3 shows partitioning examples. \( D \) circle represents a module, and an edge represents that two modules connected by the edge interact. In our discussion we focus on the basic partitioning case where two non-overlapping sets cover the entire application: \( d \cup s = D \) and \( d \cap s = \{ \} \). Note that there are more complex cases like partitioning between a client and multiple servers and partitioning with overlap sets of modules (e.g., the same module executes at the client and the server).
Fig. 3: Examples of partitioning of an application composed of four modules when module 1 is pinned to the client and module 4 is pinned to the server. Note that even with this simple application there are four possible partitioning examples. Dynamic partitioning chooses what partitioning to use at run time among the set of available partitioning configurations.

Given the dataflow application \( D_g(C_n, C), s_i, d_{i,j} \) the mobile device properties \( \{p, \delta \} \), and the wireless network bandwidth \( B \), the partitioning problem in this study is the problem of allocating a set of \( n \) 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).

\[
\max_{a_i, b_{i,j}} T_p = \frac{1}{t_p}, \quad i, j \in \{0,1, \ldots, n+1\} \quad \text{Where} \quad \forall i \in C_n, \max_{a_i, \sum_{i,j \in C} d_{i,j}(a_i-a_j)} \quad \sum_{i,j \in C} b_{i,j}(a_i-a_j)^2 = B,
\]

\[
b_{i,j} > 0,
\]

\[
a_o = 1,
\]

\[
a_{n+1} = 1,
\]

\[
a_i = 0 \quad \text{or} \quad 1, \quad i \in \{1,2, \ldots, n\}
\]

(2)

The core variables are \( a_i \) and \( b_{i,j} \). \( a_i \) is either 0 or 1 integer, indicating the offloading decision for component \( i \). If \( a_i \) equals to 1, component \( i \) is executed on the mobile device; otherwise \( a_i = 0 \) means running on the cloud. \( b_{i,j} \) is the wireless bandwidth allocated to the channel \((i,j)\). Note that two virtual nodes, 0 and \( n+1 \), 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 \((n, n+1)\) are added into the set of edges \( C \) of the dataflow graph, where node 1 is the entry node and node \( n \) is the exit node. Accordingly, \( d_{0,1} \) is the size of a unit of input data. \( d_{n,n+1} \) is the size of a unit of output data.
III. FRAMEWORK OF EXECUTION

A. Overview

Fig. 4 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, a 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 $\delta$, and the current network bandwidth $B$. Using this dynamic information from mobile device as well as the static application properties stored in cloud, the application master then generates an optimal partitioning result, which is presented in next Section. 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 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.

![Diagram showing the relationship between the mobile client and the application master.](image)

**Fig. 5:** Relationship between the mobile client and the application master

Fig. 5 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. Taking the result as the input, the underlying module DF Execution provides 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. 6 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, 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, or two remote components, e.g., (3; 6) in graph. 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.

The objective function shown in Equation (2) depends on two variables, $a_i$ and $b_{i,j}$. 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 $X = \{ a_i \mid i = 1, 2, \ldots, n \}$, the throughput is maximized when $b_{i,j}$ satisfies the condition that

![Dataflow Execution of Distributed system](image)

**Fig. 6:** Dataflow Execution of Distributed system
\[
\begin{align*}
&b_{ij} = \frac{d_{ij}}{t_{\text{comm}}(X)} \quad \forall(i,j) \in C \quad \text{and} \quad a_i \neq a_j \\
&b_{ij} = 0 \quad \forall(i,j) \in C \quad \text{and} \quad a_i = a_j \\
&b_{0,1} = (1 - a_i) \frac{d_{0,1}}{t_{\text{comm}}(X)} , \\
&b_{n,n+1} = (1 - a_i) \frac{d_{n,n+1}}{t_{\text{comm}}(X)} , \quad \ldots \ldots \quad (3)
\end{align*}
\]

Where \(t_{\text{comm}}(X)\) is the communication cost/time that each crossing channel needs to transfer a unit of data.

\[
t_{\text{comm}}(X) = \frac{1}{B} \left[ (1 - a_i) S_{i,j} + \sum_{j \in C} d_{j} (a_i - a_j)^2 \right] , \quad \ldots \ldots \quad (4)
\]

So, the original problem can be reduced into

\[
\max_X T = \frac{1}{\max\{ t_{\text{comp}}(X), t_{\text{comm}}(X) \}} \quad \ldots \ldots \quad (5)
\]

Where \(t_{\text{comp}}(X) = \max_{a_i \in X} \alpha_i \cdot S_i \cdot \sum_{i \in C} a_i \) 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.

**IV. EVALUATION**

The performance metric we consider in the evaluation is the throughput of the data stream application. First, we evaluate how the controlling parameters affect the performance. Second, we study the effect of the input parameters including application graphs, the wireless networking bandwidth \(B\) and the available computing resource at mobile device \(\delta p\). At last, we demonstrate the factor that can affect the computational cost. 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 [11]. 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 a computation-intensive application.

\[
\text{CCR} = \frac{[d_{0,1} + d_{n,n+1} + \sum_{j \in C} d_{i,j}] / [(e + 2) \times B]}{[\sum_{i \in C} S_i] / (n \times \delta p)} \quad \ldots \ldots \quad (6)
\]

We have done a group of experiments to evaluate the effect of both controlling parameters and input parameters to the performance.
V. CONCLUSIONS

In this paper, we argue for dynamic partitioning of applications between weak device and cloud to better support applications running in diverse devices in different environments. We formalize the dynamic partitioning problem and sketch how to construct a system that supports dynamic partitioning. We then 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. We believe that dynamic partitioning is an important part of future mobile cloud computing.

In future, we are going to conduct a series of experimental tests on real-world applications such as hand gesture recognition and mobile augmented reality based on our proposed framework. 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 two strategies like with/without partitioning.

REFERENCES