Proceedings of

The FTRA 2014 International Symposium on Frontier and Innovation in Future Computing and Communications

FCC 2014

January 13 – 16

Auckland, New Zealand
Optical computing for digital signal process in incoherent fiber system using positive realization

Kyungsup Kim, Jaecheol Ryou, and Woo-Tak Jung

Optimization of Non-Fuzzy Neural Networks Based on Crisp Rules in Scatter Partition

Keon-Jun Park, Byun-Gon Kim, Kwan-Woong Kim, Jung-Won Choi and Yong-Kab Kim

LBG-based Non-Fuzzy Inference System for Nonlinear Process

Keon-Jun Park, Kwan-Woong Kim, Byun-Gon Kim, Jung-Won Choi and Yong-Kab Kim

An Efficient Embedded System for the Detection and Recognition of Speed-Limit Signs

Hsing-Lung Chen, Ming-Sze Chen, Shu-Hua Hu

Future Enterprise Computing Looking Into 2020

Yung-Chin Fang, Yuxiang Gao, Cindy Stap

Implementation of a Multi-Core Prototyping System for a Video-based Fire Detection Algorithm

Yong-Hun Park, Myeongsu Kang, Jong-Myon Kim

A Novel Spectrum Sensing Scheme based on Wavelet Denoising with Fuzzy for Cognitive Radio Sensor Networks

Hiep Vu-Van and Insoo Koo

Accelerating a Bellman-Ford Routing Algorithm Using GPU

In-Kyu Jeong, Jia Uddin, Myeongsu Kang, Cheol-Hong Kim, Jong-Myon Kim

Early Fire Detection Using Multi-Stage Pattern Recognition Techniques in Video Sequences

Dongkoo Shon, Myeongsu Kang, Junsang Seo, Jong-Myon Kim

Energy Efficient Backoff-Free Fragment Retransmission Fragmentation Scheme for IEEE 802.11 WLAN

Prosper Mafole, Yuki Manabe, Teruaki Kitasuka, Masayoshi Aritsugi

Fault Classification of an Induction Motor Using Texture Features of Vibration Signals

Won-Chul Jang, Myeongsu Kang, Jong-Myon Kim Kim

An Efficient Audit Scheme for Cloud Storage

Ohmin Kwon, Dongyoung Koo, Yongjoo Shin, and Hyunsoo Yoon

High-Performance Cluster Estimation Using Many-Core Models

Junsang Seo, Myeongsu Kang, Cheol-Hong Kim, Jong-Myon Kim

Dynamic Processing of Floating Continuous Query for Cold Chain Management

Bonghee Hong, Kyungmin Lee, and Yongxu Piao

A New Approach for Task Scheduling Optimization in Mobile Cloud Computing

Pham Phuoc Hung, Tuan-Anh Bui, Eui-Nam Huh
A New Approach for Task Scheduling Optimization in Mobile Cloud Computing

Pham Phuoc Hung¹, Tuan-Anh Bui², Eui-Nam Huh¹

¹Department of Computer Engineering, Kyung Hee University, Yongin-si, South Korea
{hungpham, johnhuh}@khu.ac.kr
²Louvain School of Engineering, Catholic University of Louvain, Louvain-la-Neuve, Belgium
tuan.bui@student.uclouvain.be

Abstract. Mobile cloud computing is growing rapidly because its device (i.e., smart phone) is becoming one of the main processing devices for users nowadays. However, there are still some negative impacts that affect cloud access, especially when access to cloud becomes expensive but recent studies are not yet efficient in eliminating these. In this paper, we present an effective task scheduling by collaborating thick-thin clients and cloud to guarantee a better accessibility to cloud network and boost up the processing time in the mobile cloud platform while considering the network bandwidth and cost for cloud service usage. Intensive simulation proves that our method can improve the task scheduling efficiency and is better cost-effective than other works.

Keywords: Task scheduling, offloading, parallel computing, thin-thick client.

1 Introduction

The last decade has witnessed a revolution in the way people access to the Internet thanks to the incredibly fast proliferation of smart devices and latest advancements in mobile communication network. Personal and business tasks are no longer attached to heavy, bulky desktop or laptop computers, etc (known as thick clients), but can use much smaller and thinner devices (known as thin clients) like smartphones or tablets, or the Google glass[1], iWatch[12]. Switching work to mobile devices does not come without problems though. Small memory, weak CPU power and short battery life or unpredictable network connectivity [2] are some of the inherent obstacles that prevent mobile devices from satisfying the increasingly sophisticated applications demanded by users. Meanwhile, gaining more and more popularity in recent years, Cloud Computing (CC) offers a considerable complement to mobile terminals by expanding their power and enabling them to perform far beyond their original capabilities. This convergence results in a new paradigm named mobile cloud computing (MCC) [4, 5] allowing mobile devices to run computation-intensive applications, providing end users with rich computing experience.

Much as MCC can enrich mobile end users’ experience, it however introduces some additional headaches related to cloud access. Researchers have been working hard to
make MCC more accessible, especially when access path to cloud network becomes unavailable or too expensive. In [7], authors argue that smart objects (phones, tablets) can exploit computing resources from nearby nodes to achieve better cloud access. These efforts, unfortunately, are not sufficient to eliminate the above discussed problems of thin clients. Thick clients, including powerful smartphones (multi-cores, big RAM, LTE-enabled), on the other hand, usually come with better hardware and network connectivity. It is understandably suggested that thin clients be coupled with thick clients to achieve a desirable access to cloud, as stated in [8]. This paper aims at extending the work in [8] to utilize the thin-thick client collaboration and cloud in task scheduling to reduce the processing time of the system in the mobile cloud platform. We particularly consider both network contention and finance of cloud users. Simulation results prove that our model has better tradeoff between cost efficiency and workflow execution time than other existing work.

The remainder of this paper is structured as follows: Section 2 presents some related work which have partly solved the discussed problems. Section 3 gives the motivating scenario in which the role of thin-thick client collaboration becomes crucial. System architecture is presented in section 4. Section 5 details the problem formulation. And section 6 specifies the implementation, performance evaluation. The last section concludes our paper and suggests the future work.

2 Related Work

There have been numerous studies which attempt to solve task scheduling problems. In [3], authors propose a task scheduling approach for assigning processors to task graph templates prepared in advance. The limitation of this method is not to consider the network contention. Sinnen et al. [6] present an efficient task scheduling method based on network contention which, however, does not ponder on monetary cost paid by cloud customers (CCs) for using the resources.

In heterogeneous CC environment, despite numerous efforts, task scheduling remains one of the most challenging problems [9]. Authors in [10] introduce a cost-efficient approach to select the most proper system (private or public cloud) to execute the workflow according to deadline constraint as well as cost saving. J. Li et al. in [15] present a scheduling algorithm to schedule the application of large graph processing in consideration of both cost and schedule length. Research literature has not yet presented, though, any scheduling approach that concerns both execution time, network contention and the cost for using cloud services in combination of thin-thick clients and clouds. In this paper we try to solve the above shortage by introducing a method that can find an optimal schedule while keeping in mind the aforementioned constraints.

3 Motivating Scenario

The scenario illustrated in Figure 1 exemplifies the utilization of thin-thick client collaboration in optimizing task scheduling: A Western girl is visiting a museum in South Korea and she enjoys seeing things that reflect Korean history and tries to understand
more about it. Much of the information there, unfortunately, is written in Korean, which causes the girl unable to understand. As she is reading, she uses her voice to order her Google glass she is wearing to take pictures of the content in Korean language and uploads them to the cloud. While the Google glasses itself does not have a good Internet connection (because its bandwidth is too low), it is configured to get connected with the girl’s powerful smart phone (in her pocket), which supports LTE connection. The pictures are automatically transferred to the phone which can then upload them to the Internet in a relatively short time, thanks to superb connection speed of LTE network. Once uploaded to the cloud, these pictures are quickly processed to extract text (using some optical character recognition technology) and translate it into the girl’s preferred language (e.g. English). Translated content is then returned to the phone before transferring to the glass then the girl can read and understand it now.

![Fig. 1. Motivating Scenario](image)

4 System Architecture

The following section gives an insight of our system architecture to address issues discussed above: Our architecture has two layers, as illustrated in Figure 2, including (1) Cloud Provider layer, which contains Virtual Machines (VMs), and (2) Cloud Customer layer, where thin clients and thick clients reside. In the second layer, there is a thick client $m$ functioning as a centralized management node, also known as a broker, which (1) receives all computation requests of users, (2) manages processor’s profiles (processing capacity, network bandwidth) as well as computation costs together with results of data query returned from processors, and (3) accordingly creates the most reasonable schedule for an input workflow. Especially, it sends data to clouds in a single connection but when VMs send data to cloud customer layer, the data will be divided into different parts with different sizes before being delivered to thick clients in multi connections according to a previous research [8]. Moreover, the system has to satisfy the following requirements:

- A STUN information and a communication library can be shared by these P2P and thick clients to thin clients or vice versa.
- Thick clients should store a copy of persistent data of the cloud, and should keep this loosely synchronized
5 Problem Formulation

In this part, we first define the terms used and then formulate the problem. Eventually, we present a method to solve the above problems.

**Definition 1.** A task graph (e.g. shown in Figure 3) is presented by a Directed Acyclic Graph $G = (V, E, w, c)$ where the set of vertices $V = \{v_1, v_2, ..., v_k\}$ represents the set of parallel subtasks, and the directed edge $e_{ij} = (v_i, v_j) \in E$ presents the communication between subtasks $v_i$ and $v_j$, $w(v_i)$ associated with task $v_i \in V$ represents its computation time and $c(e_{ij})$ represents its communication time between task $v_i$ and task $v_j$. We presume that a task $v$ without any predecessors, $prec(v) = 0$, is called entry task $v_{en}$ and the task that does not have any successors, $succ(v) = 0$ is named end task $v_{en}$. Each task $v_i$ has a different priority because some tasks should be serviced earlier than other ones [11]. It consists of workload $w_i$ which delimits amount of work processed at the computing resources. Besides, it also contains the set of preceding subtasks $prec(i)$, the set of successive subtasks $succ(i)$ of task $i$, $t_s(v_i,P)$ denotes Start Time and $w(v_i,P)$ means the Execution Time of task $v_i\in V$ on processor $P$. Hence, the finish time of that task is given by $t_f(v_i,P) = t_s(v_i,P) + w(v_i,P)$. Suppose that the following conditions are satisfied:

**Condition 1.** A task cannot begin its execution until all its inputs have been gathered sufficiently.

**Condition 2.** The available time $avail(v_j,P)$ is the time that processor $P$ completes the last assigned task and be ready to execute task $v_j$. Therefore,

$$ avail(v_j, P) = \max_{e_{ij} \in E, v_i = pred(v_j)} (t_f(e_{ij})) $$ (1)
Condition 3. Let $[A,B] \in [0, \infty]$ be an idle time interval on processor $P$, an interval in which no node is executed. A free task $v_i \in V$ can be scheduled on processor $P$ within $[A,B]$ if $\max \{ A, \text{avail}(v_i, P) \} + w(v_i, P) \leq B$ (2)

![Fig. 3. A sample DAG](image)

Definition 2. A processor graph $TG=(N,D,H)$ is a graph that describes the topology of a network between vertices (processors) which are VMs, thick or thin clients. In this model, $N$ is the finite set of vertices, a directed edge $d_{ij} \in D$ denotes a directed link from vertex $n_i$ to vertex $n_j$, $n_i, n_j \in N$, $H$ is a finite set of hyper edges representing a multi-directional link between vertices. In this graph, each processor $i$ controls processing rate $p_i$ and bandwidth $b_{wi}$ to communicate with other processors.

Definition 3. Task scheduling $S$ of a DAG $G=(V,E,w,c)$ on a target system having network topology $TG$ is to assign processor nodes of that system to task nodes of DAG in order to minimize total execution time. The input of task scheduling is a task graph and process graph. And the output is a schedule which is an assignment of a processor to each task node.

Proposed Method
In this part, we make some following assumptions for our proposed method. Given a task graph $G = (V,E,w,c)$ and the processor graph with network topology $TG=(N,D,H)$, our approach, extended from the Contention Aware Scheduling (CAS) algorithm [6], has two steps:

(a) Determining the task priority to make the order of the tasks.

In this step, each task is set a priority based on the upward rank value of this task in the task graph. Here, a priority of a task $v_i$ is estimated by the length of the critical path leaving the task. Recursively defined, the priority value $pr$ of a task $v_i$ is as follow:

$$pr(v_i)=\begin{cases} \bar{w}(v_i)+\max_{v_j \in \text{success}(v_i)} [\bar{c}(e_{ij})+pr(v_j)] & v_i \neq \text{end task} \\ \bar{w}(v_i) & v_i = \text{end task} \end{cases}$$ (3)

where $\bar{w}(v_i)$ is the average computation time of task $v_i$ and $\bar{c}(e_{ij})$ is the average data transfer time between task $v_i$ and $v_j$, correspondingly:

$$\bar{w}(v_i) = \frac{\sum_{n_i=0}^{w_i} p_i}{n} \quad , \quad \bar{c}(e_{ij}) = \frac{\sum_{n_j=0}^{b_{wj}}}{n}$$ (4)

with $n$ is the number of processors in the cloud environment.

Let $c(e_{ij})$ be the data transfer time from processor $n_j$ to processor $n_k$ to execute task $i$, then $c(e_{ij})$ is defined as following:

$$c(e_{ij}) = (d_{ij}^i + \sum_{n_k \in \text{prev}(j), n_k \in \text{success}(j)} ad_{nk}^j) * \left( \frac{1}{bw_j} + \frac{1}{bw_k} \right)$$ (5)

...
In the formula (5), $d_i^j$ is the amount of input data stored at processor $P_j$ and used for executing task $v_i$ and $ad_i^j$ is amount of outgoing data from $P_j$ to process task $v_i$.

Finally, we sort all tasks with a descending order of $pr$, which is the length of remained schedule.

(b) Choosing the most appropriate processor to execute the selected task.

Once all preceding tasks of $v_i$ are completed, the start time of a task presents the time when the last preceding task of $v_i$ is completed. Hence, to determine that start time, the earliest idle interval $[A, B]$ on processor $P$ has to be searched to satisfy condition 3. As a result, the start time of task $v_i$ is set as:

$$t_s(v_i, P) = \begin{cases} 0 & v_i = v_{entry\_task} \\ \max(A,\text{avail}(v_i, P)) & v_i \neq v_{entry\_task} \end{cases} \quad (6)$$

Thus, the Earliest Start Time ($EST$), and Earliest Finish Time ($EFT$) values of a task $v_i$ executed on a processors $P$ are computed as follow:

$$EST(v_i, P) = \max_{v_j \in \text{prev}(v_i), v_k \in N_i} (t_j(v_i, P)) + \max(c_i(t_k^j)) \quad (7)$$

$$EFT(v_i, P) = w(v_i, P) + EST(v_i, P) \quad (8)$$

Besides, the algorithm also considers the cost paid by cloud customers for using cloud resources that are used to execute the tasks. The cost $C(v_i, P)$ for task $v_i$ executed at a VM $P_j$ or at a thin client $P_j$ as well as thick client $P_j$ is defined by.

$$C(v_i, P) = \begin{cases} C_{\text{process}}(v_i, P) + C_{\text{wait}}(v_i, P) + C_{\text{transfer}}(v_i, P) + C_{\text{connect}}(v_i, P) & \text{for VMs} \\ C_{\text{transfer}}(v_i, P) + C_{\text{connect}}(v_i, P) & \text{for thin-thick clients} \end{cases} \quad (9)$$

In the formula (9), each cost is calculated as following:

Cost of processing is expressed as:

$$C_{\text{process}}(v_i, P) = c_1 \cdot \frac{w_i}{p_j} \quad (10)$$

where $c_1$ is the processing cost per a time unit of workflow execution on processor $P_j$ with processing rate $p_j$.

Let $t_{\text{fin}}$ be the finishing time of the task which is completed first out of the parallel tasks and there is no available task after this one, $c_2$ is the waiting cost per time unit, $t_i$ is the finish time of the $i^{th}$ task then the cost of waiting time is as:

$$C_{\text{wait}}(v_i, P) = c_2 \cdot (t_i - t_{\text{fin}}) \quad (11)$$

Suppose that the amount of money per transferring time unit of outgoing data unit from processor $P_j$ is $c_3$, then the cost of transferring time is defined as follow:

$$C_{\text{transfer}}(v_i, P) = c_3 \cdot (d_i^j + \sum_{\text{prev}(v_i), \text{meets}(j)} ad_i^j) / bw_j \quad (12)$$

We assume that the distribution of disconnection event is a Poisson distribution with parameter $\mu_T$. $\mu_T$ is determined by the stability of the network. The expected number of arrivals over an interval of length $\tau$ is $E[N_T] = \mu_T \cdot \tau$. Let $L$ be the random variable of the length of an offline event, $\mu_L$ is the expected length or mean of length and $c_4$ is disconnection cost per unit time. Therefore, the expected duration time of disconnection event which can affect to the processing time of task $v_i$ is $\mu_T \cdot \tau \cdot \mu_L$. Hence, the cost of disconnection is as:

$$C_{\text{disconnect}}(v_i, P) = c_4 \cdot (\mu_T \cdot \tau \cdot \mu_L) \quad (13)$$
From this cost, we can calculate the utility function which computes the tradeoff between the cost and $EFT$ as formula (14).

$$\text{Min} \sum_{v \in V, P \in P} \left( \frac{\text{cost}(v, P)}{\text{Max}\{\text{cost}(v, P)\}} \times \frac{EFT(v, P)}{\text{Max}\{EFT(v, P)\}} \right)$$

(14)

By considering the above utility function, we can determine the most appropriate processor $P$ which processes task $v_i$ is the one whose combination between $\text{cost}(v_i, P)$ and $EFT(v_i, P)$ should acquire the minimum value.

6 Implementation and Analysis

This section presents our experiments via numerical simulations to evaluate the efficiency of our approach and compare its performance with two others: CAS [6], which just takes account of network contention, and Classical Heuristic Scheduling (CHS), which merely concerns $EFT$. All the parameters are different task graphs $G=(V, E, w, c)$ with the increase of the matrix size from 20 to 100 and heterogeneous processor graphs $TG=(N,D,H)$ which is a combination between 15VMs with the different configurations and 5 thick clients and 4 thin clients located at the local system of CCs for the above algorithms. We developed the simulations in Java with jdk-7u7-i586 and Netbeans-7.2. In the following figures, it is obvious to see that there are some differences between the simulated results. On the left of the Figure 4 shows that CHS get the worst case on schedule length, CAS obtains the best result while our approach is still in the middle. Specifically, our method is 18% better than CHS. However, on the right of the Figure 4, according to the monetary cost paid by CCs, it can be seen that although CAS provides the best performance, it takes the biggest cost. In the meantime, CHS approach spends the lowest cost while our solution gets balanced between schedule length and cloud cost. Compared with CAS, our method can save nearly 22% cost for CCs.

![Fig. 4. Comparison our approach with others about schedule length and cost.](image)

![Fig. 5. Schedule length and Cost](image)
Besides, we measured the effect of the increasing number of processors on the cloud cost and the schedule length while fixing the quantity of tasks. The result is reflected in Figure 5 which indicates that the more number of processors is used, the better performance our system gets but the cost CCs have to pay is higher as well.

7 Conclusion

This paper discusses an architecture based on the joint work of thin-thick clients and cloud in optimizing the task scheduling in mobile cloud platform. Our work especially tries to bring desired processing time while balancing the network contention and cloud service cost in order to bring a cost-effective solution, which appears to be better than other existing approaches when compared with. We will soon extend the proposed model to run in a various circumstances to achieve higher reliability and better performance.

Acknowledgement. This research was supported by the MSIP (Ministry of Science, ICT&Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (NIPA-2013-H0301-13-4006) supervised by the NIPA(National IT Industry Promotion Agency). The corresponding author is Eui-Nam Huh.

References