Joint Optimization of Task Mapping and Routing for Service Provisioning in Distributed Datacenters

Huawei Huang, Deze Zeng, Song Guo
School of Computer Science and Engineering,
University of Aizu, Aizu-wakamatsu, Japan
sguo@u-aizu.ac.jp

Hong Yao
School of Computer,
China University of Geosciences, Wuhan, China
yaohong@cug.edu.cn

Abstract—Service provisioning has been widely regarded as a critical issue to quality-of-service (QoS) of cloud services in datacenters. Conventional studies on service provisioning mainly focus on task mapping, i.e., how to distribute the service-oriented tasks onto the servers to achieve different goals, e.g., makespan minimization. In distributed datacenters, a task is usually routed from its generation point (i.e., control room) to the designated server within a datacenter network. Since the routing delay also has a deep influence on the task makespan, we are motivated to study how to minimize the maximum makespan of all tasks in a duty period by joint optimization of both task mapping and routing. It is formulated as an integer programming with quadratic constraints (IPQC) problem and proved as NP-hard. To tackle the computational complexity of solving IPQC, a heuristic algorithm with polynomial time is proposed. Extensive simulation results show that it performs close to the optimal one and outperforms existing algorithms significantly.

I. INTRODUCTION

Cloud computing emerges as a new computing paradigm that enables end users to acquire various services hosted in datacenters according to their requirements, without awareness of where the services are executed and how they are managed. All these details are transparent to users and managed by Internet Services Providers (ISPs) with the help of virtualization techniques in cloud computing platforms, e.g., Yahoo! data centers, Amazon Elastic Cloud (EC2), Microsoft Windows Azure platform and Google App Engine, etc.

One critical issue that is most concerned by cloud computing service users is the user experience or the quality-of-service (QoS). A commonly adopted metric is makespan, which describes how fast a request can be responded. A representative cloud computing service type known as urgent service (e.g., requests submitted to searching engine, etc.) usually requires fast response, e.g., low makespan. One of the main factors that affects the service makespan is task mapping about which server in the datacenters shall be designated for its related computing task. Servers are usually with heterogeneous and capability-limited resources while the service oriented tasks impose different resource requirements such as computation, storage and communication resources. Intuitively, if many tasks are submitted to the same server, long waiting time, and thus long turnaround time on a server, may be experienced by these tasks. Therefore, it is essential to carefully make the task mapping decision. A few pioneering investigations on request provisioning have been made in the literature, e.g., [1]–[6].

Practically, datacenters are organized in a distributed way, as shown in Fig. 1. In such distributed datacenters, besides task mapping, it is also required to route a task onto its designated server, i.e., task routing. For example, datacenter 0 in Fig. 1, as a control room, may accept a lot of service requests during a duty period. It first generates corresponding tasks and then delivers them to their respectively designated servers for processing by the datacenter network. During such process, task routing is another unavoidable factor affecting the service makespan. However, existing studies usually overlook such factor and only focus on the task mapping issue. Therefore, in this paper, we are motivated to study the service provisioning problem on how to minimize the maximum makespan of all tasks in a duty period with joint optimization of both factors. The main contributions of this paper are as follows:

- We study the service provisioning for maximum task makespan minimization with joint consideration of task mapping and routing and formulate it as an integer programming with quadratic constraints (IPQC) problem.
- Since the formulated service provisioning problem is known as NP-hard, to tackle the computation complexity, we further propose a polynomial-time heuristic algorithm.
- Via extensive simulations, the performance results show the high efficiency of our algorithm by the fact that it much approaches the optimal one and substantially outperforms some existing request provisioning algorithms.

The remainder of this paper is structured as follows. Section
II reviews some related work in services provisioning. Section III introduces the system model and problem statement. Section IV formally proposes the IPQc formulation. Section V presents our polynomial heuristic algorithm. Section VI gives the evaluation results. Finally, section VII concludes this paper.

II. RELATED WORK

Virtualization is a key technique to improve QoS in cloud computing [7]. Many problems have been studied in this field. For instance, the resources allocation or resource provisioning for virtual services [1]–[6], efficient scheduling for real time jobs in virtual network [8]–[13], etc.

The processing ability, memory capacity, and communication resources are considered as three main dimensions of resource provisioning in traditional cloud. Numbers of previous related work present their solutions ensuring that server resources shall be allocated efficiently, or with lowest operation cost to achieve best profit. For example, Kim et al. [1] investigate power aware provisioning of virtual machines for real-time services. They provide approach to model such a service as a real-time virtual machine request, then provision virtual machines using DVFS (Dynamic Voltage Frequency Scaling) schemes and propose algorithms to reduce power consumption. Later on, Sun et al. [4] formulate the problem of optimal provisioning for elastic service oriented virtual network request as a mixed integer programming model which aims to improve performance and resource efficiency and to maximize the total profit of infrastructure provider. Recently, Niu et al. [5] and Rochman et al. [6] study the problem of resource allocation and utility maximization in modern large-scale network environments. They both provide theoretical analysis and algorithms to place resources in systems aiming at minimizing the cost of providing the demands from end users.

On the other hand, jobs scheduling for cloud computing also attracts many attentions. For instance, in order to validate the benefit of the Intelligent Transportation Systems (ITS)-Cloud a load balancing study is performed by Bitam et al. [9]. With the service offered by cloud to schedule job tasks, this study compares the performance conducted in ITS-Cloud and conventional cloud. Recently, Maguluri et al. [13] consider a stochastic model of jobs arriving at a cloud data center, in which the job sizes are also modeled as random variables. These jobs need to be scheduled non preemptively on servers.

The most related work is [12], in which authors integrate virtual machine migration and the traffic engineering as an integrated control with the objective of minimizing the average link delay. The considered common problem is the traffic engineering while provisioning the service requests. Distinctively, we jointly optimize the mapping and routing problems for each task. Such optimization also provides the best routing path for each task.

III. SYSTEM MODEL AND PROBLEM STATEMENT

We consider a network graph $\mathbb{G}=(\mathbb{V}, \mathbb{E})$ consisting of a datacenter set $\mathbb{V} = \{D_1, D_2, \ldots, D_\beta\}$ and a link set $\mathbb{E}$. There are $\gamma$ servers in total, denoted as $\mathbb{S} = \{s_1, s_2, \ldots, s_\gamma\}$. Each datacenter owns a certain number of servers. The deployment of servers is described by a $\gamma \times \beta$ binary matrix $\mathcal{M}^{SD} = \{M_{ij}^{SD}\}$, $i = 1, 2, \ldots, \gamma$, $j = 1, 2, \ldots, \beta$, where

$$M_{ij}^{SD} = \begin{cases} 1, & \text{if } s_i \text{ locates at } D_j, \\ 0, & \text{otherwise}. \end{cases} \quad (1)$$

Let $l_{uv}$ denote the delay on the link $(u,v)$ from $D_u$ to $D_v$ for transmitting one unit of data. The edge set $\mathbb{E}$ therefore can be denoted by a $\beta \times \beta$ matrix $\mathcal{M}^{E}$ as

$$\mathcal{M}^{E} = \{l_{uv}\}, \forall (u, v) \in \mathbb{E}. \quad (2)$$

Specially, we have $l_{uv} = 0, \forall u = v$. Furthermore, we assume that all links between datacenters are also capacity-limited. Let a $\beta \times \beta$ matrix $\mathcal{M}^{CAP}$ denote the capacities of all links. Each element $M_{uv}^{CAP} \geq 0$ represents the traffic capacity of link $(u,v) \in \mathbb{E}$ in current duty period. Obviously, $\mathcal{M}^{E}$ and $\mathcal{M}^{SD}$ are determined once the distributed datacenters are deployed while $\mathcal{M}^{CAP}$ can be estimated at the beginning of a duty period according to the datacenter traffic statistics.

Without loss of generality, we assume that all the servers in the distributed datacenters are homogeneous. The processing speed of a server is $P_0$. Each server has a set of capacity-limited resources (e.g., CPU, memory, I/O, etc.) denoted by a $\theta$-set: $\mathbb{R} = \{R_1, R_2, \ldots, R_\theta\}$. Then, the current available resources on all servers can be described by an $\theta \times \gamma$ matrix $\mathcal{A}^{RS} = \{A_{kj}^{RS}\}, k = 1, \ldots, \theta, j = 1, \ldots, \gamma$, where $A_{kj}^{RS}$ refers to the available capacity of resource $R_k$ on server $s_j$.

In a certain duty period, many requests are received at the control room $D_\alpha$, which then generates the related tasks $\mathbb{T} = \{T_1, T_2, \ldots, T_n\}$ and estimates the resource requirements of these tasks. Specially, let $T_i \in \mathbb{T}$ also represent the task size. Both the storage resource and computation resource requirements are proportional to the task size. We mainly consider the urgent cloud services with hard deadline $\zeta$. The resource requirements of all tasks are denoted by a $\theta \times \alpha$ matrix $\mathcal{Q}^{RT} = \{Q_{ki}^{RT}\}, k = 1, 2, \ldots, \theta, i = 1, 2, \ldots, \alpha$, where $Q_{ki}^{RT}$ is the amount of resource $R_k$ requested by task $T_i$. Therefore, $\mathcal{A}^{RS}$ and $\mathcal{Q}^{RT}$ can be known when tasks are generated at the control room in a duty period.

We assume that all tasks are executed on a FIFS (First-In-First-be-Served) manner in each duty period on all servers. If a task is distributed to an idle server, its turnaround time may not be processed by a server in the same datacenter with the designated server via the datacenter network. The routing delay from the control room to the designated server shall be also considered as part of the task makespan. The two factors are highly correlated with each other and therefore shall be investigated jointly.
IV. PROBLEM FORMULATION

In this section, we formulate the service provisioning problem with the joint consideration of task-server mapping and routing.

A. Task Mapping

Task Completion Constraints: Let an $\alpha \times \gamma$ binary matrix $M^{TS} = \{M_{ij}^{TS}\}$, $i = 1, 2, \cdots, \alpha$, $j = 1, 2, \cdots, \gamma$ represent the task-server mapping relation, which is defined as:

$$M_{ij}^{TS} = \begin{cases} 1, & \text{if } T_i \text{ is mapped onto server } s_j, \\ 0, & \text{otherwise}. \end{cases}$$ (3)

The task completion constraints require that each task $T_i$ must be distributed onto exactly one server, i.e.,

$$\sum_{j=1}^{\gamma} M_{ij}^{TS} = 1, \forall i = 1, 2, \cdots, \alpha.$$ (4)

Resource Capacity Constraints: A server is resource capacity-limited while the tasks assigned are attached with different resource requirements. For each type of resource, the total requirement shall not exceed the provision capacity. Then, we have:

$$\sum_{i=1}^{\alpha} Q_{k}^{RT} \cdot M_{ij}^{TS} \leq A_{k}^{RS}, \forall k = 1, \cdots, \theta, \forall j = 1, \cdots, \gamma.$$ (5)

Computation time: The actual computation time $e_i$ experienced by a task $T_i$ is only related to its size, regardless of which server it is distributed to. Then, we have:

$$e_i = \frac{T_{i}}{T_{0}}, \forall i = 1, 2, \cdots, \alpha.$$ (6)

Waiting time: Assume that there are $N_0$ existing unfinished tasks in all servers at the start of the duty period. We can use an $N_0 \times \gamma$ binary matrix $\overline{M}^{TS} = \{\overline{M}_{ij}^{TS}\}$, $i = 1, 2, \cdots, N_0$, $j = 1, 2, \cdots, \gamma$ to represent the initial server-task status, where

$$\overline{M}_{ij}^{TS} = \begin{cases} 1, & \text{old task } \overline{T}_i \text{ has been deployed on server } s_j, \\ 0, & \text{otherwise.} \end{cases}$$ (7)

Without loss generality, we assume that all existing tasks are randomly and uniformly distributed in all servers. Let $\phi(j)$ denote the computing time of all existing tasks in server $s_j$. We have:

$$\phi(j) = \sum_{i=1}^{N_0} (\overline{M}_{ij}^{TS} \cdot \overline{T}_{i} / T_0), \forall j = 1, 2, \cdots, \gamma.$$ (8)

Then, the expected maximum waiting time for any new task mapped onto server $s_j$ till its completion can be written as:

$$\varphi(j) = \phi(j) + \sum_{i=1}^{\alpha} (M_{ij}^{TS} \cdot e_i), \forall j = 1, 2, \cdots, \gamma.$$ (9)

Thus, the expected maximum turnaround time for a new task $T_i$ on a server, denoted by $\omega_i$, can be calculated as

$$\omega_i = \sum_{j=1}^{\gamma} (M_{ij}^{TS} \cdot \varphi(j)), \forall i = 1, 2, \cdots, \alpha.$$ (10)

B. Task Routing

The routing path for a task $T_i$ from the control room to its designated server is constituted with multiple successive links belonging to edges set $E$.

We first introduce a binary variable indicating whether a link $(u, v) \in E$ is adopted for routing task $i$ or not as:

$$\lambda_{uv}^i = \begin{cases} 1, & \text{if the link } (u, v) \text{ is adopted by } T_i; \\ 0, & \text{otherwise}. \end{cases}$$ (11)

A task issued from control room $D_c$ may be distributed to the server within the same datacenter or to a server in another datacenter. For the former case, no inter-datacenter link will be adopted, while one link must be used for the latter case. Therefore, we have:

$$\sum_{\forall (c, u) \in E} \lambda_{cu}^i \leq 1.$$ (12)

With the task-server mapping matrix $M^{TS}$, we can derive the task-datacenter mapping relationship as $M^{TD} = M^{TS} \times M^{SD}$, which shall be an $\alpha \times \beta$ matrix. For each $T_i$, at least one link ended with its designated datacenter must be adopted. The tasks issued from control room $D_c$ will be distributed to the server within the same datacenter or to a server in other datacenters. For the former case, no inter-datacenter link will be adopted, while one link must be used for the latter case. Therefore, we have:

$$\sum_{\forall (u, v) \in E} \lambda_{uv}^i \geq 1, \forall i = 1, \cdots, \alpha.$$ (13)

Furthermore, the total traffic for all tasks in each link must be limited within its capacity, i.e.,

$$\sum_{i=1}^{\alpha} T_i \cdot \lambda_{uv}^i \leq M_{uv}^{cap}, \forall (u, v) \in E.$$ (14)

The total routing delay for a task can be obtained by summing up all the link delays it shall experience during its routing path as:

$$r_i = \sum_{\forall (u, v) \in E} T_i \cdot l_{uv} \cdot \lambda_{uv}^i, \forall i = 1, 2, \cdots, \alpha.$$ (15)

For urgent cloud computing services, we shall ensure that the maximum task makespan does not exceed the required deadline to guarantee the QoS requirement. Therefore, we have:

$$\omega_i + r_i \leq \zeta, \forall i = 1, 2, \cdots, \alpha.$$ (16)

C. An IPQC Formulation

By summarizing all the above constraints together, we obtain an integer linear programming with quadratic constraints as:

$$\begin{align*}
\min_{M^{TS}, \lambda_{uv}} & : \sum_{i=1}^{\alpha} \omega_i + r_i \\
\text{s.t. :} & \quad (4), (5) \text{ and (12)-(16).}
\end{align*}$$ (17)

Within these constraints, (5) and (13) belong to quadratic ones. Note that the task mapping routing towards minimum span of all tasks in a duty period are jointly considered in the formulation above. The joint consideration can be observed from constraint (13).
summarized in Alg. 1 and Alg. 2.

Two-Phase heuristic, which includes a best-fit-increasing tasks packing) and the 2) traffic management problem (shortest path finding in graph). Therefore, we design a deadline-constrained Two-Phase heuristic, which includes a best-fit-increasing tasks mapping algorithm (BFI) and an integer programming based path-finding-algorithm (IP-PFA). The details of algorithms are summarized in Alg. 1 and Alg. 2.

Algorithm 1 Best-Fit-Increasing Tasks Mapping Algorithm

Input: network graph $\mathcal{G}=(\mathcal{V}, \mathcal{E})$, new tasks set $\mathcal{T}$, $\mathcal{A}^{RS}$, $\mathcal{Q}^{RT}$, $\mathcal{M}^{TS}$

Output: The mapping matrix of new tasks $\mathcal{M}^{TS}$

1: $\mathcal{M}^{TS} \leftarrow \emptyset$
2: $\mathcal{T}' \leftarrow$ Sort all new tasks by their sizes increasingly
3: $\mathcal{S}' \leftarrow$ Sort all servers by the number of their hosting tasks increasingly
4: for all $T_i \in \mathcal{T}'$, $\forall i = 1, 2, \cdots, \alpha$ do
5: for all $s_j \in \mathcal{S}'$, $\forall j = 1, 2, \cdots, \gamma$ do
6: if $s_j$ can satisfy the resource requirement of $T_i$ then
7: $M^TS_{ij} \leftarrow 1$
8: $\mathcal{S}' \leftarrow$ Re-sort all servers increasingly by the number of hosting tasks
9: Break
10: end if
11: end for
12: end for
13: $\omega_i \leftarrow \sum_{j=1}^{\gamma} (M^TS_{ij} \cdot \varphi(j)), \forall i = 1, 2, \cdots, \alpha$; s.t. (8), (9).

In Alg. 1, the mapping matrix $\mathcal{M}^{TS}$ is initialized with 0 firstly. Then, all new tasks under scheduled are sorted increasingly by their sizes and recorded in list $\mathcal{T}'$ (line 2). Later on, all servers are sorted in an increasing order of the total number of tasks they are hosting, and recorded in list $\mathcal{S}'$ (line 3). Then, each $T_i$ in list $\mathcal{T}'$ shall be mapped to a best host server $s_j$ which hosts the fewest number of tasks in its serving queue. Note that, before mapping $T_i$ to $s_j$, the resources requirement of $T_i$ must be satisfied. Once a new task is mapped onto a server, the list $\mathcal{S}'$ needs to be updated immediately (line 8). With the input arguments, Alg.1 gives tasks mapping matrix at last. Note that, the time complexity from line 4 to line 12 in Alg.1 is $O(\alpha \cdot \gamma)$. Since the number of servers is a constant once the network topology is given, the complexity of the BFI heuristic algorithm can be rewritten as $O(\gamma \cdot N)$.

Since the cloud network in practice is in a scale with at most only a few hundreds of datacenters, we adopt the integer programming based algorithm 2 as our shortest path finding scheme in the proposed Two-Phase heuristic.

Algorithm 2 IP based Path-Finding-Algorithm

Input: network graph $\mathcal{G}=(\mathcal{V}, \mathcal{E})$, tasks set $\mathcal{T}$, $\mathcal{M}^{SD}$, $\mathcal{M}^{CAP}$, $\mathcal{M}^{CE}$, and $\mathcal{M}^{TS}$ retrieved by tasks mapping algorithm.

Output: The routing path of each new task $\lambda^T_{uv}$

1: $\mathcal{M}^{TD} \leftarrow \mathcal{M}^{TS} \times \mathcal{M}^{SD}$
2: $\lambda^T_{uv} \leftarrow \emptyset$
3: $\lambda^T_{uv} \leftarrow \arg\min \sum_{i=1}^{\alpha} \sum_{(u,v) \in E}(T_i \cdot l_{uv} \cdot \lambda^T_{uv}), \text{s.t. (12), (13), (14), (16).}$

In Alg. 2, taking the obtained tasks mapping result as one of the input arguments, we firstly analyze the mapping matrix $\mathcal{M}^{TD}$ (line 1). Then, the optimal routing path for each task can be retrieved at last (line 3). Additionally, the expected maximum delay by invoking Two-Phase heuristic must be within the desired deadline according to the following two operations. First, record the expected maximum turnaround time for each new task $T_i$ after mapping them to corresponding servers (line 13 in Alg. 1). Second, the routing delay of $T_i$ is restricted within the required deadline via constraint (16), which is shown in the line 3 of Alg. 2.

V. HEURISTIC ALGORITHM

Our urgent service provisioning problem with constraints on link and resource capacity is known as NP-hard [4], [14]. It actually consists of: 1) the task-server mapping problem (bin-packing) and the 2) traffic management problem (shortest path finding in graph). Therefore, we design a deadline-constrained Two-Phase heuristic, which includes a best-fit-increasing tasks mapping algorithm (BFI) and an integer programming based path-finding-algorithm (IP-PFA). The details of algorithms are summarized in Alg. 1 and Alg. 2.

In order to compare the performance on tasks mapping with proposed Two-Phase heuristic, without loss of generality, we consider the size of each task equal to the memory size it
consumes in simulations. The following bin-packing heuristics are also considered for the purpose of comparison.

**FFD-MEM [16]:** First-fit-decreasing (FFD) places items in a decreasing order of size, and at each step, the next item is placed to the first available bin. **FFD-MEM** is the FFD solution sorted by memory requirements.

**BFD-MEM [16]:** Best-fit-decreasing (BFD) places a virtual task in the fullest server that still has enough capacity. **BFD-MEM** represents the BFD solution sorted by memory requirements.

However, in order to fit our evaluation, we adjust these two heuristics slightly by enforcing the deadline constraint.

### A. Expected Maximum Makespan in Small-Scale Network

We firstly study the sum of expected makespan of all new tasks in this set of simulation. Without loss of generality, we normalize the capacity of all links in network, i.e., $M_{uv}^{cap}$, into a common value in all sets of simulation involved in traffic management.

Since the expected makespan of each task includes two terms mentioned previously, in this set of simulations, we assign the maximum makespan as the delivery delay generated by **IP-PFA** algorithm plus the maximum waiting time caused by **BFI**, **FFD-MEM** and **BFD-MEM**, respectively.

Setting $\beta, \gamma, \zeta, l_{uv}$ as 6, 12, 450, 1, respectively, we vary $\alpha$ from 4 to 12 and fix capacity of link as 1000 to study the effect of number of new tasks. From Fig. 2(a), the linear increasing trend can be observed from all the three curves.

Secondly, by varying the capacity of link from 800 to 1600, we see from Fig. 2(b) that the sum of expected maximum makespan of all new tasks slightly decreases over the capacity of link. This is attributed to the fact that when enlarging link capacity, more tasks find their shorter individual paths.

At last, we could see the sum of maximum makespan as an increasing function of the probability of existing old tasks in a server from 0.6 to 1 as shown in Fig. 2(c). This is because the larger of this probability indicates more old tasks in servers, resulting in longer waiting time for new tasks. Additionally, the performance tends to be more stable, because tasks have been mapped onto the few servers near to the control room. In other words, there will be little increment on expected waiting time or delivering delay when such probability increases.

### B. Expected Maximum Makespan in Large-Scale Network

We use the ITALYNET [4] (Fig.1) with 21 datacenter nodes and 35 links in this set of simulation. The requests are generated randomly and uniformly such that the number of new emerging tasks in a certain duty-period is equal to a given number $\alpha$. Without loss of generality, we assign control room in datacenter $D_0$.

We distribute 210 servers evenly in datacenters. There are 50 new tasks need to be scheduled in current duty-period. The probability of existing old tasks and the maximum number of old tasks queuing in each server are set as 0.6 and 5, respectively. Furthermore, the deadline, unit of delay on links and capacity of links are set to 500, 1 and 10000, respectively.
Fig. 4. Sum of maximum makespan v.s. the capacity of link.

Fig. 5. Sum of maximum makespan v.s. probability of existing old tasks in a server.

In the traffic management module, the classical and widely used shortest path algorithms, i.e., Dijkstra [17] and SPF A [10], are adapted under the link capacity constraints. The resulting algorithms denoted by Dijk.CAP and SPFA.CAP, respectively, are compared to Alg. 2 in terms of the sum of maximum makespan of all tasks.

We first study the effect of workload in the network by varying the number of new tasks from 50 to 250. In Fig. 3, we observe that the sum of expected maximum makespan of all tasks using all the schemes are linearly increasing with the workload. Specially, from Fig. 3(d) we see the Two-Phase heuristic performs the best within the three combined schemes. Fig.4 clearly shows the sum makespan of all new tasks as a decreasing function of the inter-datacenter link capacity from 4000 to 20000. The reason is similar to the small-scale scenarios discussed in Section VI-A. Similarly, via varying the probability of existing old tasks in a server within \([0.6, 1.0]\), we finally observe that the sum of expected maximum maskspan increases over this probability from Fig. 5. The proposed Two-Phase heuristic performs better than the other two schemes again.

VII. CONCLUSION

In this paper, we study the service provisioning problem in distributed datacenters. Conventional studies on service provisioning only focus on task mapping. In distributed datacenters, a task shall be routed from control room to the designated server within a datacenter network making the routing delay critical to the makespan as well. In this paper, we are dedicated to addressing how to minimize the maximum makespan of all the tasks in a duty period by joint optimization of both task mapping and routing, which is then formulated as an integer programming with quadratical constraints problem. We also propose a heuristic algorithm. Its high efficiency is verified by simulations in both small and large-scale networks.

REFERENCES


