

QoS-Aware Optimization of Composite-Service Fulfillment Policy

Chun Zhang, Rong N. Chang, Chang-Shing Perng, Edward So, Chunqiang Tang, Tao Tao
IBM T.J. Watson Research Center
{czhang1, rong, perng, edwardso, ctang, ttao}@us.ibm.com

Abstract—In a service-oriented IT infrastructure, functional capabilities of a computing component are externalized via one or more *service* interfaces. Driven by the demand for business agility and return-on-investment optimization, various dynamic service discovery and composition technologies have been proposed and developed with a common goal of enabling business-aligned fulfillment of customer requests. However, from the viewpoint of capacity planning and IT optimization, much work is still needed in helping an enterprise decide the “optimal” IT resources necessary for deploying the atomic services in support of those composite ones. The service deployment decision must be integrated with the request fulfillment policy so that the differentiated quality-of-service (QoS) requirements of service requests can be met, for instance, with minimum hardware/software cost.

In this paper, we propose an approach for QoS-aware optimization of composite-service fulfillment policy. Without loss of generality, we assume that the optimization goal is to minimize the number of machines subject to response time and throughput requirements. After presenting our approach to the optimization problem using the assumption, we show that an NP-hard throughput optimization problem must be attacked. We then illustrate how we attack the problem via an efficient heuristic algorithm. The algorithm decomposes the end-to-end response time requirement for each type of composite service into atomic-service level response time assurance, and co-locate atomic services with similar response time assurance on machines with similar utilization characteristics. The algorithm exemplifies an integrated approach to optimizing service deployment and service composition. We demonstrate that the algorithm achieves a substantially higher throughput than a common baseline algorithm.

I. INTRODUCTION

In an IT infrastructure built according to Service Oriented Architecture (SOA) principles [1], functional capabilities of a computing component are externalized via one or more “service” interfaces such as WSDL-specified Web Services. Driven by the demand for business agility and return-on-investment optimization, various dynamic service discovery and composition technologies have been proposed with a common goal of enabling business-aligned fulfillment of customer service requests [2]–[7]. Workflow technologies, for example, can dynamically build a composite service from atomic services through the use of control flows such as sequential, branch, parallel and loop.

However, from the viewpoint of capacity planning and IT optimization, much work is still needed in helping an enterprise decide the “optimal” number of machines needed to deploy the necessary atomic services in support of those composite ones. The service deployment decision must be integrated with the end-to-end customer-request fulfillment policy so that the differentiated quality-of-service (QoS) requirements for

composite service requests can be met, for example, with the minimum infrastructure/hardware/software/management cost.

As IT resources are limited, it is important to allocate them wisely in order to maximize business values. Ultimately, the attained business performance depends upon both the volume (i.e., throughput) of completed customer requests and the satisfaction of the customers in terms of request response time. The balance between throughput and response time needs to be considered not only at the atomic-service level (as common practice in capacity planning and IT optimization), but also at the composite-service level. After all, what matters is the customer-perceived *end-to-end* QoS.

In this paper, we propose an approach to QoS-aware optimization of composite-service fulfillment policies. Without loss of generality, we assume that the optimization goal is to minimize the number of machines subject to customer response time and throughput requirements. We will show that one key step to achieve this goal is to maximize throughput while observing IT resources and service response time constraints. The challenge of the throughput-maximization problem mainly stems from the difficulty in decomposing the end-to-end composite-service level response time requirement into atomic-service level ones. Based on our insight into the round-robin CPU scheduling policy [8] that most systems use, we propose an efficient algorithm that decomposes the composite-service level response time requirements into atomic-service level response time requirements that are proportional to the CPU consumption of the atomic services. To satisfy the derived response time requirements at the atomic-service level, the algorithm further groups together atomic services with similar QoS requirements and co-locates them on machines with similar utilization characteristics.

Studies on fulfillment policy to deliver composite-service level QoS guarantee are quite limited. Recently, [9] considered the problem with the assumption that the CPU execution time of an atomic service instance is inversely proportional to the number of allocated machines. However, this assumption requires perfect parallel speed up in case of multiple allocated machines — which may be difficult to achieve due to non-ignorable parallel overhead [10]. In this paper, we consider the problem with alternative assumption that there is no parallel speed up in case of multiple allocated machines.

The remainder of this paper is organized as follows. Section II introduces our SOA system model and related key concepts of service deployment and service composition. In Section III, we formulate an integrated service deployment and service composition optimization problem to satisfy customer requests with QoS requirements. In Section IV, we prove the formu-

lated problem is NP-hard, and propose a heuristic algorithm. Section V evaluates the proposed heuristic algorithm. In Section VI, we conclude this paper with a summary and some discussion of this work.

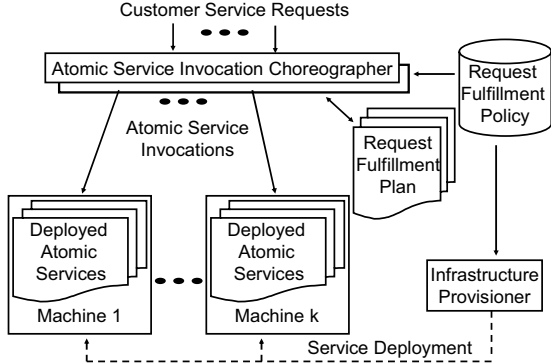


Fig. 1. Our system model.

II. SOLUTION OVERVIEW

This section presents our system model, and some basic concepts of service deployment and service composition.

A. System Model

Figure 1 depicts our system model. Our integrated optimizer intelligently computes the *request fulfillment policy*. This policy guides the *infrastructure provisioner* to allocate IT resources and deploy atomic services with certain functional and QoS properties. When a customer request arrives, the *atomic service invocation choreographer* effectively generates the *request fulfillment plan* according to a request fulfillment policy and then executes it by invoking the underlying atomic services. Note that the request fulfillment policy created by our *integrated optimizer* determines *both* service deployment and service composition. These concepts are explained below.

B. Concepts of Atomic Service Deployment

An *abstract atomic service* implements certain functional requirements, e.g., credit card authorization or warehouse catalog searching. After an atomic service is deployed to a machine with certain IT resources allocated to it, this *deployed atomic service* possesses certain QoS characteristics, e.g., the number of customer requests that it can process per second (throughput) and the average response time to complete the request. Note that the same abstract atomic service can be deployed to different machines, and these deployed atomic services may exist concurrently and provide different QoS characteristics.

Figure 2 shows an example of five atomic services s_1 — s_5 . Atomic service s_3 runs on two machines m_3 and m_5 , but the two deployments provide different QoS guarantees, i.e., ($t = 2.5/s, r = 0.1s$) and ($t = 2.5/s, r = 1s$), where $t = 2.5/s$ means processing 2.5 requests per second, and $r = 0.1s$ means the average response time is 0.1 seconds.

We allow different atomic services to share a single machine (see atomic services s_3 and s_5 on machine m_5 in

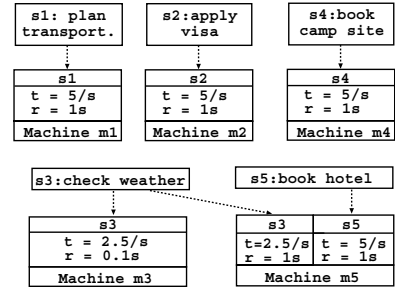


Fig. 2. An example of five atomic services.

Figure 2), and assume that machines use the standard round-robin scheduling policy [11], as shown in Figure 3. A newly arriving request joins a single, shared queue regardless of the atomic service it targets. A request waits in the queue in a first-come-first-service (FCFS) fashion, and is allocated an infinitesimal quantum $\delta \rightarrow 0$ of CPU time when it reaches the head of the queue. When that quantum expires and if processing this request needs more time, the request is added back to the tail of the queue. ($\delta \rightarrow 0$ is an ideal case where the CPU resource is evenly consumed by all queued requests at any time).

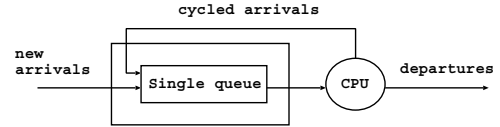


Fig. 3. The round-robin scheduling policy.

C. Concepts of Service Composition

Customer requests are of different *types*. For a given type, a composite-service template specifies the abstract atomic services involved in processing requests of this type and the composition rules that connect the atomic services into a complete end-to-end service. A composite-service template can be represented as a Directed Acyclic Graph (DAG) of atomic services, where a high-level composite service is built from low-level atomic services using sequential, parallel, and branch compositions. An example of a composite-service template is shown in Figure 4.

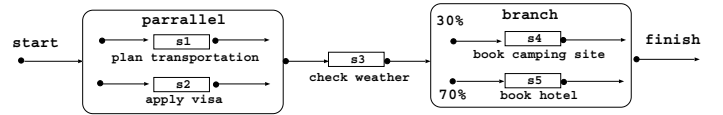


Fig. 4. The trip-planning composite-service template.

In addition to the composite-service template, customer requests of a given type also come with certain QoS requirements such as throughput and average response time. Our algorithm recursively decomposes these end-to-end QoS requirements into the requirements at the atomic-service level. The details will be explained later. Internally, our algorithm builds one or more execution plans for processing customer requests of a given type. When a request of this type arrives,

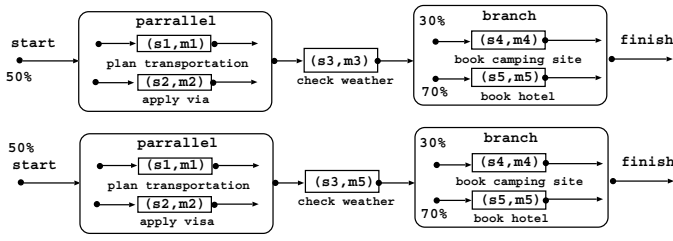


Fig. 5. Two execution plans of the trip-planning service. See Figure 2 for the QoS characteristics of the deployed atomic services.

the system chooses one of the plans with certain probability to execute the request.

An execution plan concretely maps the abstract atomic services in a composite-service template to the deployed atomic services with certain QoS guarantees so that the end-to-end QoS requirements of the customer requests are satisfied. Figure 5 shows two execution plans of the trip-planning composite-service. For customer requests with QoS requirements ($t = 5/s, R = 3s$), each of the two execution plans is chosen with probability 0.5.

D. Our Approach

Below, we outline the high-level steps of our approach to QoS-aware optimization of a composite-service fulfillment policy. Without loss of generality, we assume that the optimization goal is to minimize the number of machines subject to response time and throughput requirements.

- 1) Infer the arrival rates of various customer requests. This requires historical workload data collected from a running IT system.
- 2) Infer the average CPU cycles needed to process one request for a given atomic service, which can be done by applying the data regression method [12] to historical data about machine CPU utilization and customer request arrival rate.
- 3) Determine composite-service templates and their internal parameters such as the branch-taken ratio for branch composition. This requires historical data on invocations of atomic services by composite services, which can be collected from a running system by monitoring tools.
- 4) Collect from service contracts the QoS requirements for each type of customer request.
- 5) Determine the minimum number of machines needed to meet the customer requirements by doing a binary search on the number of configured machines. Given a fixed number of available machines, they are sufficient for the need if the maximum throughput they can support subject to response time constraints is higher than or equal to the throughput specified in the service contract.

The last step, solving the throughput maximization problem, is key to our approach. As known solutions exist for the other steps, this paper focuses on this last step. We will prove that the throughput maximization problem is NP-hard, and show an effective heuristic solution.

III. THROUGHPUT MAXIMIZATION PROBLEM FORMULATION

In this section, we formulate the integrated service deployment and service composition problem. We first introduce the notations (see Table I), and then formulate the problem. Finally, we discuss the difference between our integrated approach and the alternative that treats service deployment and service composition separately.

TABLE I
NOTATIONS

β	total system throughput
$\omega(y_l)$	probability of type y_l customer request
Γ	mapping assured to deployed atomic services
$\Phi(y_l)$	type y_l customer atomic service visit ratio
A	QoS-assured atomic service set
$B(y_l)$	type y_l customer branch fraction set
C	machine CPU capacity
D	QoS-assured atomic service capabilities
$h(p) = (t(p), r(p))$	QoS of execution plan p
M	machine set
N	atomic service CPU consumption
$P(y_l)$	type y_l customer request execution plan set
$p'(y_l)$	type y_l QoS assured execution plan
Q	deployed atomic service capabilities
r, R	average response time and its QoS requirement
S	atomic service set
t, T	throughput and its QoS requirement
U	machine CPU load
V	atomic service deployment set
$W(y_l) = (S(y_l), E(y_l))$	type y customer composite-service template
Y	customer type set

A. Notations for Service Deployment

Atomic service: $S = \{s_1, s_2, \dots, s_{|S|}\}$, where s_i denotes the i -th atomic service.

Atomic service required CPU: $N = \{n_{s_1}, n_{s_2}, \dots, n_{s_{|S|}}\}$, where n_{s_i} denotes the CPU cycles needed to process one request for atomic service $s_i \in S$.

Machine: $M = \{m_1, m_2, \dots, m_{|M|}\}$, where m_k denotes the k -th machine. Each machine could be a physical machine, or a virtual machine implemented by VMware [13].

Machine CPU capacity: C . We assume that each machine's CPU capacity is C , measured in cycles per second.

Atomic service deployment: $V = \{(s_i, m_k)\}$, $s_i \in S$, $m_k \in M$, where (s_i, m_k) denotes the deployment of atomic service s_i on machine m_k , simplified as $v_{i,k}$.

Atomic service deployment QoS: $Q = [q(v_{i,k})]$, $s_i \in S$, $m_k \in M$, where $q(v_{i,k}) = (t(v_{i,k}), r(v_{i,k}))$ denotes the throughput $t(v_{i,k})$ and average response time $r(v_{i,k})$ of deployed atomic service $v_{i,k}$.

Machine CPU load: $U = \{u_{m_1}, u_{m_2}, \dots, u_{m_{|M|}}\}$, where u_{m_k} denotes the CPU load of machine m_k . We have

$$u_{m_k} = \sum_{v_{i,k} \in V} n_{s_i} t(v_{i,k}) \leq C. \quad (1)$$

Suppose machines use the round-robin scheduling policy, and the arrival of requests for an atomic service running on one machine is a Poisson process. It has been shown [8]

that, for $M/G/1$ queue using the round robin policy, the average response time is a convex and increasing function of the machine CPU load, i.e.,

$$r(v_{i,k}) = \frac{n_{s_i}}{C - u_{m_k}} = \frac{n_{s_i}}{C(1 - \rho_{m_k})}, \quad (2)$$

where $\rho_{m_k} = u_{m_k}/C$ denotes the utilization of machine m_k .

B. Notations for Service Composition

Customer request type: $Y = \{y_1, y_2, \dots, y_{|Y|}\}$, where y_l denote the l -th customer request type. $S(y_l)$ denotes the set of atomic services used in processing type y_l customer requests.

Customer request type probability: $\Omega = \{\omega(y_1), \omega(y_2), \dots, \omega(y_{|Y|})\}$, where $\omega(y_l)$ denotes the probability that a request is of type y_l .

Customer request response time requirement: $R = \{R(y_1), R(y_2), \dots, R(y_{|Y|})\}$, where $R(y_l)$ denotes the response time requirement of type $y_l \in Y$ customer requests.

Customer request throughput: $t = \{t(y_1), t(y_2), \dots, t(y_{|Y|})\}$, where $t(y_l)$ denotes the throughput of type $y_l \in Y$ customer requests. We have

$$\forall y_l, y_m \in Y, \quad \frac{t(y_l)}{t(y_m)} = \frac{\omega(y_l)}{\omega(y_m)}. \quad (3)$$

Total system throughput β :

$$\beta = \sum_{y_l \in Y} t(y_l). \quad (4)$$

Composite-service template: $W = \{W(y_1), W(y_2), \dots, W(y_{|Y|})\}$, where $W(y_l) = \{S(y_l), E(y_l)\}$, $S(y_l) \subseteq S$, is a directed acyclic graph representation of the composite-service template for type $y_l \in Y$ customer requests. The links in $E(y_l)$ are denoted by $(s_i, s_j) \in S \times S$.

Branch fraction: $B = \{b(y_1), b(y_2), \dots, b(y_{|Y|})\}$, where $b(y_l) = \{b_{s_i, s_j}(y_l)\}$, $(s_i, s_j) \in E(y_l)$. $b_{s_i, s_j}(y_l) \in b(y_l)$ denotes the branch fraction from atomic service s_i to s_j . For sequential and parallel composition between atomic services s_i and s_j , we always have $b_{s_i, s_j}(y) = 1$.

Visit ratio: $\Phi = \{\phi(y_1), \phi(y_2), \dots, \phi(y_{|Y|})\}$, where $\phi(y_l) = \{\phi_{s_i}(y_l)\}$, $s_i \in S(y_l)$ denotes the average number of visits to s_i per type y_l customer request. If s_i is the starting state, we have $\phi_{s_i}(y_l) = 1$. Otherwise, we have

$$\phi_{s_i}(y_l) = \frac{\sum_{(s_j, s_i) \in E(y_l)} \phi_{s_j}(y_l) b_{s_j, s_i}(y_l)}{z_{s_i}(y_l)}, \quad (5)$$

where $z_{s_i}(y_l)$ is the number of parallel branches merged at atomic service s_i in composite-service template $W(y_l)$, $y_l \in Y$. $z_{s_i}(y_l) = 1$ if s_i is not a merging point of parallel paths in composite-service template $y_l \in Y$.

Execution plan: $P = \{P(y_1), P(y_2), \dots, P(y_{|Y|})\}$, where $P(y_l)$ denotes the set of execution plans for type y_l customer request. An execution plan $p \in P(y_l)$ is a vector of mappings $p = [p(s_i)]$, $s_i \in S(y_l)$, where $p(s_i)$ maps an abstract atomic service $s_i \in S(y_l)$ to one of its deployed atomic services $v_{i,k} \in V$ running on machine k .

Execution plan QoS: $h(p) = (t(p), r(p))$, $p \in P(y_l)$, $y_l \in Y$, where $t(p)$ and $r(p)$ are throughput and average response time of the execution plan p , respectively.

The average response time $r(p)$ of execution plan $p \in P(y_l)$ is calculated from the response time of the building-block atomic services in a bottom-up fashion, based on the compositions (i.e., sequential, branch, and parallel) specified in the service template $W(y_l)$. Suppose a higher-level block is composed from a set of low-level sub-blocks. Let \tilde{r} be the block average response time, and r_i the i -th sub-block average response time. For sequential composition, we have

$$\tilde{r} = \sum_i r_i, \quad (\text{sequential composition}). \quad (6)$$

For parallel composition, we have

$$\tilde{r} = \max_i r_i, \quad (\text{parallel composition}). \quad (7)$$

For branch composition, the block average response time is the weighted sum of sub-block average response time. We have

$$\tilde{r} = \sum_i b_i r_i, \quad (\text{branch composition}), \quad (8)$$

where b_i is the branch ratio of sub-block i .

The total throughput of the execution plans for type y_l customer requests is $\sum_{p \in P(y_l)} t(p)$.

$$t(y_l) = \sum_{p \in P(y_l)} t(p). \quad (9)$$

The throughput of execution plan, $t(p)$, $p \in P(y)$ is set to 0 if its average response time does not meet the customer requirement, i.e.,

$$t(p) = 0 \text{ if } r(p) > R(y_l), p \in P(y_l). \quad (10)$$

The throughput of a deployed atomic service $v_{i,k}$ is the total throughput of all execution plans using it,

$$t(v_{i,k}) = \sum_{y_l \in Y} \sum_{p \in P(y_l)} \mathbf{1}(p(s_i) = v_{i,k}) t(p) \phi_{s_i}(y_l). \quad (11)$$

Here $\mathbf{1}(X)=1$ if the predicate X is true; otherwise, $\mathbf{1}(X)=0$.

C. Problem Formulation for Integrated Service Deployment and Service Composition

The integrated service deployment and service composition problem is formulated as follows.

Integrated problem for service deployment and service composition.

Inputs: (1) atomic services S with per-request CPU consumption N ; (2) machines M with CPU capacity C ; (3) customer request type Y with probability Ω , template W , and average response time requirement R .

Maximize: the total system throughput β

Subject to: Equations (1)-(11). That is, there exist deployed atomic services V and execution plans P that can satisfy customer QoS requirements R .

Let β^* denote the maximum system throughput derived by solving above formulated problem.

D. Difference between integrated and decoupled approaches

Unlike our integrated solution, an alternative approach may treat service deployment and service composition separately. With this decoupled approach, the service composition optimizer first decomposes the end-to-end QoS requirements of customer requests into atomic-service level requirements, and then the service deployment optimizer allocates CPU resources to meet these atomic-service level QoS requirements.

We use the example in Figure 6 to demonstrate the difference between the integrated and decoupled approaches. In this example, the composite service consists of two sequential atomic services. Figure 6(a) shows the decoupled approach, where the end-to-end customer QoS requirements are $R = 1s$ which means the average response time should be no longer than 1 second.

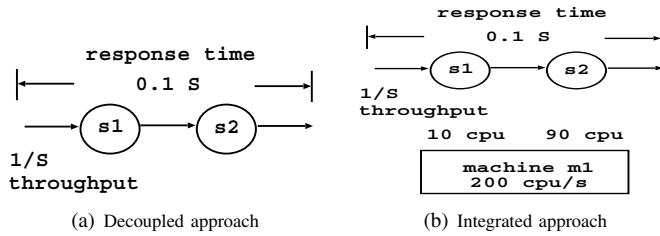


Fig. 6. An example to illustrate the difference between decoupled and integrated service deployment and service composition.

In order to meet the response time requirement, we have

$$\forall p \in P, t(p) > 0, \quad r(p(s_1)) + r(p(s_2)) = 1s. \quad (12)$$

However, from the service composition point of view, it is unclear how to decide $r(p(s_1))$ and $r(p(s_2))$. For instance, both $(r(p(s_1)), r(p(s_2))) = (0.5s, 0.5s)$ and $(r(p(s_1)), r(p(s_2))) = (0.1s, 0.9s)$ satisfy equation (12).

Our integrated approach solves this problem by leveraging extra information such as CPU scheduling policy, the CPU consumption of atomic services, the number of machines, and the CPU capacity of machines. In Figure 6(b), the first atomic service s_1 consumes 10 CPU cycles in order to process one request, while s_2 consumes 90 CPU cycles per request. All atomic services are deployed on a single machine with CPU capacity 200 using the round-robin scheduling policy. With these information, we know from eq. (2) that the integrated approach picks $(r(p(s_1)), r(p(s_2))) = (0.1s, 0.9s)$ since all other combinations can not be satisfied by service deployment.

IV. INTEGRATED SERVICE DEPLOYMENT AND SERVICE COMPOSITION HEURISTIC ALGORITHM

In this section, we first prove that the integrated problem formulated in Section III is NP-hard, and then propose a heuristic algorithm.

A. NP-hardness

Theorem 1: The integrated service deployment and service composition problem is NP-hard.

Proof: We prove it by a reduction from the well-known PARTITION [14] problem.

We therefore propose a heuristic algorithm. The NP-hardness mainly arises from the difficulty in decomposing the end-to-end response time requirement into the deployed atomic-service level. Based on the observation from Equation (2), which shows that the average response time is proportional to the CPU consumption, we propose a heuristic algorithm. The key idea of our algorithm is to first decompose the end-to-end average response time requirement to so-called QoS-assured atomic services in proportion to atomic-service CPU consumption, and then use deployed atomic-services to implement the QoS capabilities of QoS-assured atomic services. We start with additional notations related to QoS-assured atomic services.

B. Additional Notations Used in Our Algorithm

QoS-assured atomic services: $A = \{(s_i, y_l)\}$, $s_i \in S$, $y_l \in Y$, where (s_i, y_l) denotes the QoS-assured atomic services for type y customer request, simplified as $a_{i,l}$.

QoS-assured atomic-service capabilities: $D = [d(a_{i,l})]$, $a_{i,l} \in A$, where $d(a_{i,l}) = (t(a_{i,l}), r(a_{i,l}))$ denotes the throughput $t(a_{i,l})$ and average response time $r(a_{i,l})$ QoS capabilities of $a_{i,l}$.

QoS-assured atomic-service based execution plan: $p' = \{p'(y_1), p'(y_2), \dots, p'(y_{|Y|})\}$, where $p'(y_l)$ denotes the QoS-assured atomic-service based execution plan for type y_l customer. Specifically, $p'(y_l) = [p'(s_i)]$, $s_i \in S(y_l)$ is a vector of mappings, where $p'(s_i) = a_{i,l}$ maps atomic service $s_i \in S(y_l)$ to its unique QoS-assured atomic service $a_{i,l}$.

Mapping from QoS-assured atomic services to deployed atomic services: $\Gamma = \{\gamma(a_{i,l}, v_{i,k})\}$, $a_{i,l} \in A$, $v_{i,k} \in V$, where $\gamma(a_{i,l}, v_{i,k}) \geq 0$ refers to the throughput of (s_i, j) supported by deployment $v_{i,k}$. We have

$$\gamma(a_{i,l}, v_{i,k}) = 0 \text{ if } r(v_{i,k}) > r(a_{i,l}), \quad (13)$$

$$\sum_{v_{i,k} \in V} \gamma(a_{i,l}, v_{i,k}) = t(a_{i,l}), \quad (14)$$

$$\sum_{a_{i,l} \in A} \gamma(a_{i,l}, v_{i,k}) \leq t(v_{i,k}). \quad (15)$$

C. Our Heuristic Algorithm

Our algorithm uses binary search to identify the maximum total throughput β^* that can be supported by a set of machines. It maintains two throughput variables β_0 and β_1 . Here β_0 is an already-known feasible total throughput, and β_1 is a throughput whose feasibility is yet to be examined. If β_1 passes the feasibility test, we set $\beta_0 := \beta_1$ and $\beta_1 := 2\beta_1$; otherwise, we set $\beta_1 := (\beta_1 + \beta_0)/2$. Our algorithm stops when the difference between β_0 and β_1 is sufficiently small.

Binary Search for Maximizing Throughput:Init: $\beta_0 = 0$ and $\beta_1 = 1$.

```

while ( $\beta_1 - \beta_0 > \epsilon$ ) {
  if ( $\beta_1$  is feasible) {  $\beta_0 = \beta_1$  and  $\beta_1 = 2\beta_1$  }
  else {  $\beta_1 = (\beta_1 + \beta_0)/2$  }
}

```

Output: β_0 .**D. Feasibility Test**

Given throughput β , the feasibility test algorithm attempts to come up with a solution that can meet this throughput requirement. The feasibility test consists of two steps: QoS decomposition and service deployment. The first step decomposes the end-to-end QoS requirement into requirements at the atomic-service level. The second step allocates CPU resources to satisfy the QoS requirements of atomic services.

1) *QoS Decomposition*: For each type of composite-service template $y_l \in Y$, we create a set of QoS-assured atomic services (s_i, y_l) . The end-to-end QoS requirements (throughput and response time) are decomposed into QoS-assured atomic services as follows.

For throughput, we *exactly* calculate the throughput of each QoS-assured atomic service. Given type $y_l \in Y$ customer with throughput requirement $t(y_l)$, the throughput of QoS-assured atomic service $t(s_i, y_l)$ is

$$t(s_i, y_l) = t(y_l)\phi_{s_i}(y_l). \quad (16)$$

For response time, as a *heuristic*, we decompose the end-to-end response time requirement into atomic-service level response time requirements proportional to the CPU consumption of the atomic services. This heuristic is the only approximation step of our overall algorithm.

The bottleneck CPU requirement is calculated for each composition block (i.e., sequential, branch, and parallel) of service template $W(y_l)$ in a bottom-up fashion. Let \tilde{n} and n_i be the bottleneck CPU requirement of a high-level block and its i -th sub-block, respectively. For sequential composition, we simply have

$$\tilde{n} = \sum_i n_i \quad (\text{sequential composition}). \quad (17)$$

For parallel composition, we have

$$\tilde{n} = \max_i n_i \quad (\text{parallel composition}). \quad (18)$$

For branch composition, the high-level block requirement is the average of the sub-block requirements. We have

$$\tilde{n} = \sum_i b_i n_i \quad (\text{branch composition}), \quad (19)$$

where b_i the branch ratio of sub-block i .

Unlike the calculation of the CPU requirement, the end-to-end average response time is decomposed into the requirements for QoS-assured atomic services in a top-down fashion. Let \tilde{r} and r_i be the response time requirements of a high-level block and its i -th sub-block, respectively. For parallel

composition, the sub-block response time is equal to the block response time, because the block response time is achieved only if all sub-block response times are achieved, i.e.,

$$r_i = \tilde{r} \quad (\text{parallel composition}). \quad (20)$$

For sequential and branch compositions, the sub-block response time is proportional to its CPU consumption:

$$r_i = \frac{n_i}{\tilde{n}} \tilde{r} \quad (\text{sequential and branch composition}). \quad (21)$$

2) *Service Deployment*: The QoS decomposition step generates QoS-assured atomic-service set $A = \{(s_i, y_l)\}$, $s_i \in S$, $y_l \in Y$ with throughput and response time QoS requirements D . The service deployment step allocates CPU resources to satisfy these requirements.

We know from equation (2) that (s_i, y_l) must be deployed on machines with a CPU utilization ρ_{m_i} less than or equal to $1 - n_{s_i}/C/r(s_i, y_l)$. Let $\tilde{\rho}(s_i, y_l)$ denote the machine utilization threshold for (s_i, y_l) , we have

$$\tilde{\rho}(s_i, y_l) = 1 - n_{s_i}/C/r(s_i, y_l). \quad (22)$$

We present a greedy algorithm that *optimally* allocates CPU resources to QoS-assured atomic services $A = \{(s_i, y_l)\}$, $s_i \in S$, $y_l \in Y$ with throughput and response time guarantees D . The algorithm first considers the allocation for QoS-aware atomic services with lower CPU utilization threshold, then those with higher CPU utilization threshold.

Greedy Algorithm for Service Deployment:

Sort the QoS-assured atomic-service set A in ascending order of utilization $\tilde{\rho}$, and set the index of A to $j = 1$.

Initialize machine id $k=1$ and available CPU capacity $X=C$.

```

while (true) {
  if ( $j > |A|$ ) { Output "feasible" and quit. }
  if ( $k > |M|$ ) { Output "infeasible" and quit. }
  Let  $(s_i, y_l)$  be the  $j$ -th element of  $A$ .
  Set  $X = \min\{X, C\tilde{\rho}(s_i, y_l)\}$ .
  if ( $t(s_i, y_l)\omega_{s_i} < X$ ) {  $j++$ ;  $X - = t(s_i, y_l)\omega_{s_i}$  }
  else if ( $t(s_i, y_l)\omega_{s_i} = X$ ) {  $j++$ ;  $k++$ ;  $X = C$  }
  else {  $k++$ ;  $t(s_i, y_l)- = X/\omega_{s_i}$ ;  $X = C$  }
}

```

E. Optimality of Our Algorithm

Due to NP-hardness of the throughput optimization problem, our heuristic algorithm can not always find optimal solution. Next, we show that for a narrower problem setting (composite-service templates that only use sequential and/or branch compositions), the solution found by our heuristic algorithm $\tilde{\beta}^*$ approaches the optimal solution β^* as the number of machines approaches infinity. We demonstrate this by proving Theorem 2,3,4,5.

Theorem 2 states that for composite-service templates that only use sequential and/or branch compositions, the machine utilization threshold for all constituent atomic services are same.

Theorem 2: Considering composite-service template sets Y that only use sequential and/or branch compositions,

$$\forall y_l \in Y, \forall s_i \in S(y_l), \check{\rho}(s_i, y_l) = 1 - \frac{n(y_l)}{R(y_l)C}. \quad (23)$$

where $n(y_l)$ and $R(y_l)$ are the average CPU requirement and response time requirements of composite-service template $y_l \in Y$, respectively.

Proof: From equation (21), we have

$$n_{s_i}/r(s_i, y_l) = n(y_l)/R(y_l). \quad (24)$$

Combined with equation (22), we have

$$\forall y_l \in Y, \forall s_i \in S(y_l), \check{\rho}(s_i, y_l) = 1 - \frac{n(y_l)}{R(y_l)C}. \quad (25)$$

Theorem 2 allows to define the machine utilization threshold at composite-service template level. Define $\check{\rho}(y_l)$, $y_l \in Y$ the machine threshold for composite-service template $y_l \in Y$ with only sequential and branch compositions. We have

$$\check{\rho}(y_l) = 1 - \frac{n(y_l)}{R(y_l)C}. \quad (26)$$

Theorem 3 further expresses the throughput upper bound $\check{\beta}^*$ using above defined $\check{\rho}(y_l)$.

Theorem 3: Consider composite-service templates that only use sequential and/or branch compositions. The throughput upper bound, $\check{\beta}^*$, is

$$\check{\beta}^* = 1 / \sum_{y_l \in Y} \frac{\omega(y_l)n(y_l)}{C|M|\check{\rho}(y_l)}. \quad (27)$$

Proof: See Appendix A.

Theorem 4 states the condition that upper bound $\check{\beta}^*$ can be achieved by our heuristic algorithm.

Theorem 4: Consider composite-service templates that only use sequential and/or branch compositions. If $\forall y_l \in Y$, $\frac{\omega(y_l)n(y_l)}{C\check{\rho}(y_l)} / \sum_{y_l \in Y} \frac{\omega(y_l)n(y_l)}{C|M|\check{\rho}(y_l)}$ is integer, then the maximum throughput achieved by the heuristic algorithm $\check{\beta}^*$ is equal to optimal solution β^* .

$$\check{\beta}^* = \beta^* = \bar{\beta}^*. \quad (28)$$

Proof: The upper bound of the throughput $\check{\beta}^*$ can be achieved by assigning $\frac{\omega(y_l)n(y_l)}{C\check{\rho}(y_l)} / \sum_{y_l \in Y} \frac{\omega(y_l)n(y_l)}{C|M|\check{\rho}(y_l)}$ machines to atomic services belonging to composite-service template y_l . For each template y_l , the throughput is $\omega(y_l) / \sum_{y_l \in Y} \frac{\omega(y_l)n(y_l)}{C|M|\check{\rho}(y_l)}$. The total throughput $\check{\beta}^*$ satisfies equation (28).

The asymptotic optimum property of our heuristic algorithm is given by Theorem 5.

Theorem 5: Considering composite-service templates that only use sequential and/or branch compositions, let $\check{\beta}^*$ be optimal throughput found by our algorithm, and β^* the maximum throughput. We have

$$\lim_{|M| \rightarrow \infty} \check{\beta}^* / \beta^* = 1. \quad (29)$$

Proof: Let \mathcal{M} be the smallest integer such that $\forall y_l \in Y$, $\frac{\omega(y_l)n(y_l)}{C\check{\rho}(y_l)} / \sum_{y_l \in Y} \frac{\omega(y_l)n(y_l)}{C\mathcal{M}\check{\rho}(y_l)}$ is integer. From Theorem 4, for integer $k > 0$,

$$\bar{\beta}^*(k\mathcal{M}) = \beta^*(k\mathcal{M}) = k\beta^*(\mathcal{M}). \quad (30)$$

For $(k+1)\mathcal{M} \geq |M| > k\mathcal{M}$, we have

$$\bar{\beta}^*(|M|) \geq \bar{\beta}^*(k\mathcal{M}) = k\beta^*(\mathcal{M}), \quad (31)$$

$$\beta^*(|M|) \leq \beta^*((k+1)\mathcal{M}) = (k+1)\beta^*(\mathcal{M}). \quad (32)$$

Dividing equation (31) by (32), we get

$$\frac{\bar{\beta}^*(|M|)}{\beta^*(|M|)} \geq \frac{k}{k+1}, \quad \lim_{|M| \rightarrow \infty} \frac{\bar{\beta}^*(|M|)}{\beta^*(|M|)} = 1. \quad (33)$$

F. Complexity Analysis

Our algorithm is fast. The loop in Section IV-C runs for up to $\log \beta^*$ iterations, where β^* is the maximum possible throughput of customer requests. The complexities of the QoS decomposition step and the service deployment step are $2|S|$ and $O(|M| + |Y||S|)$, respectively. Hence, the complexity of the algorithm is $O(\log \beta^*(|M| + |Y||S| + 2|S|))$.

V. PERFORMANCE RESULTS

In this section, we evaluate the performance of our algorithm by comparing it with a baseline algorithm that considers only throughput during service deployment.

A. Baseline Algorithm for Comparison

The baseline algorithm attempts to maximize the throughput of customer requests without considering the response time requirements. It assumes that any machine can process requests for any atomic service s_i . Upon receiving a request for s_i , the baseline algorithm picks uniformly at random a machine from the machine set to process the request. Let β^* denote the optimal maximum throughput and $\hat{\beta}^*$ denote the maximum throughput of the baseline algorithm.

Theorem 6: If all processed requests are counted in throughput regardless of their response times (i.e., $\forall y_l \in Y$, $R(y_l) = \infty$), the baseline algorithm achieves the optimal maximum throughput, i.e., $\hat{\beta}^* = \beta^*$.

Proof: The proof directly follows from the fact that the baseline algorithm evenly distributes load to all machines.

However, if we count in throughput only those processed requests whose response times are satisfied, we have $\hat{\beta}^* \leq \beta^*$.

Let $\check{\beta}^*$ be the maximum throughput of our algorithm. Because our algorithm takes into account both throughput and response time, its throughput is always better than or equal to that of the baseline algorithm.

Theorem 7: $\check{\beta}^* \geq \hat{\beta}^*$.

Proof: See Appendix B.

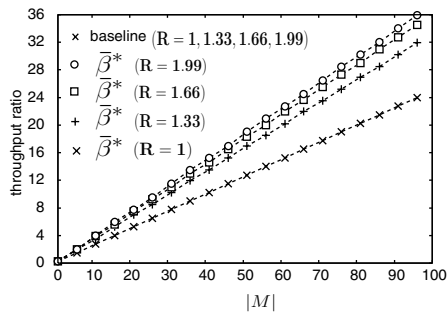


Fig. 7. Comparing the algorithms when $\omega(y_1) = \omega(y_2) = 50\%$.

B. Performance Comparison with the Baseline Algorithm

We use the following setup in the comparison. The system processes only two types of customer requests. The composite-service template of the first customer request type, $W(y_1)$, consists of a single atomic service s_1 that needs n_{s_1} CPU cycles to process one request. Similarly, the composite-service template of the second customer type $W(y_2)$ is composed of a single atomic service s_2 with CPU requirement n_{s_2} . The response time requirements are $R(y_1) = 1s$ and $R(y_2) = Rs$, respectively. In the comparison, we vary R from 1 to 1.99. We assume that a received request belongs to one of the two customer types with equal probability, i.e., $\omega(y_1) = \omega(y_2) = 50\%$.

Figure 7 shows the system throughput when $n_{s_1} = n_{s_2} = 0.8G$ CPU cycles. When $R = 1$, the baseline algorithm and our algorithm have identical performance, because the QoS requirements of the two customer types are identical. As R increases, the QoS requirement of one customer type relaxes and one would expect the throughput to improve as well. However, the throughput of the baseline algorithm remains unchanged because it treats all customers equally regardless of their QoS requirements. In contrast, our algorithm delivers a higher throughput as expected, because it uses low-utilization machines for customer requests with high QoS requirements. Compared with the baseline algorithm, when $R = 1.99$, our algorithm satisfies 50% more customer requests.

Parameters $\omega(y_1)$ and $\omega(y_2)$ represent the relative arrival rates of customer requests of different types. Figure 8 shows the throughput when $(\omega(y_1), \omega(y_2)) = (10\%, 90\%)$ and $(\omega(y_1), \omega(y_2)) = (90\%, 10\%)$, respectively. Compared with the baseline algorithm, our algorithm delivers 148% higher throughput when $(\omega(y_1), \omega(y_2)) = (10\%, 90\%)$ and $R = 1.99$, i.e., when the customer requests with low-QoS requirements dominate. In this case, our algorithm allocates a small number of machines to serve customer requests that require fast response, while the majority of the machines are used to serve customer requests that are not stringent on response time.

VI. CONCLUSION

This paper studies the problem of QoS-aware optimization of composite-service fulfillment policy. We demonstrate that utilizing the cross-layer relationship data in optimization (i.e., the relationship between composite services, atomic services, and machines) help achieve a better performance. Without loss of generality, we assume that the optimization goal is to

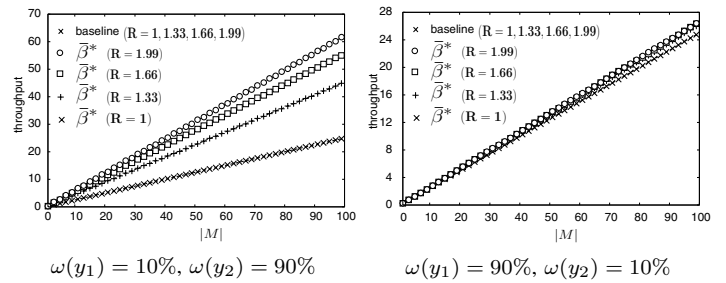


Fig. 8. Comparing the algorithms when varying $\omega(y_1)$ and $\omega(y_2)$.

minimize the number of machines subject to response time and throughput requirements. As a key step toward this goal, we formalized and solved the throughput maximization problem.

We proved that this problem is NP-hard, and proposed an efficient algorithm that maximizes the end-to-end service throughput while observing response time constraints. We further proved that the solution of our algorithm is always better than or equal to that of a baseline algorithm that considers only throughput during service deployment. Empirically, we demonstrated that under the same response time requirements, our algorithm achieves a substantially higher throughput than the baseline algorithm. This outstanding performance is due to our novel optimization techniques. First, we decompose the end-to-end service response time requirement into atomic-service level response time requirements that are proportional to the CPU consumption of the atomic services. Second, during service deployment, we group together atomic services with similar QoS requirements and co-locate them on machines with similar utilization characteristics.

REFERENCES

- [1] "IBM service oriented architecture," <http://www.ibm.com/soa>.
- [2] X. Gu, K. Nahrstedt, R. Chang, and C. Ward, "Qos-assured service composition in managed service overlay networks," in *Proceedings of IEEE 23rd ICDCS*, 2003.
- [3] L. Zeng, B. Benattallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.
- [4] D. Ardagna and B. Pernici, "Global and local qos constraints guarantee in web service selection," in *Proceedings of IEEE International Conference on Web Services*, 2005.
- [5] M. J. Buco, R. N. Chang, L. Z. Luan, E. So, C. Tang, and C. Ward, "Pem: A framework enabling continual optimization of workflow process executions based upon business value metrics," in *Proceedings of the 2005 IEEE International Conference on Services Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 33–40.
- [6] D. Gannon, B. Plale, M. Christie, L. Fang, Y. Huang, S. Jensen, G. Kandaswamy, S. Marru, S. L. Pallickara, S. Shirasuna, Y. Simmhan, A. Slominski, and Y. Sun, "Service oriented architectures for science gateways on grid systems," in *ICSOC*, 2005, pp. 21–32.
- [7] K. Verma, P. Doshi, K. Gomadam, J. Miller, and A. Sheth, "Optimal adaptation in web processes with coordination constraints," in *Proceedings of the IEEE International Conference on Web Services*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 257–264.
- [8] L. Kleinrock, *Queueing Systems*. John Wiley And Sons, 1976.
- [9] W. Lin, Z. Liu, C. H. Xia, and L. Zhang, "Optimal capacity allocation for web systems with end-to-end delay guarantees," *Perform. Eval.*, vol. 62, no. 1–4, pp. 400–416, 2005.
- [10] M. J. Quinn, *Parallel computing (2nd ed.): theory and practice*. New York, NY, USA: McGraw-Hill, Inc., 1994.
- [11] A. S. Tanenbaum, *Modern Operating Systems (2nd ed.)*. Prentice-Hall, Inc., 2001.

- [12] S. Uttamchandani, L. Yin, G. A. Alvarez, J. Palmer, and G. A. Agha, "Chameleon: A self-evolving, fully-adaptive resource arbitrator for storage systems." in *USENIX Annual Technical Conference*, 2005.
- [13] "VMware," <http://www.vmware.com/>.
- [14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.
- [15] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

VII. APPENDIX A

Proof of Theorem 3:

Let start with additional notations.

Required CPU cycle rates by template y_l : $f(y_l)$, $y_l \in Y$.

We have

$$f(y_l) = t(y_l)n(y_l). \quad (34)$$

Required CPU cycle rates of machine m_k **by template** y_l : $f(y_l, m_k)$, $y_l \in Y$, $m_k \in M$. We have

$$f(y_l, m_k) = \sum_{s_i \in S(y_l)} \mathbf{1}(p(s_i) = v_{i,j})t(p)\phi_{s_i}(y_l)n_{s_i}. \quad (35)$$

Note that $f(y_l) = \sum_{m_k \in M} f(y_l, m_k)$.

Utilized fraction of machine m_k **by template** y_l :

$$x(y_l, m_k) = f(y_l, m_k)/u_{m_k}.$$

Total utilized machine fraction by template y_l : $X(y_l) = \sum_{m_k \in M} x(y_l, m_k)$. We have

$$\sum_{y_l \in Y} X(y_l) = |M|. \quad (36)$$

Utilization over total utilized machine fraction by template y_l : $g(y_l) = f(y_l)/X(y_l)$.

We prove Theorem 3 through two lemmas.

Lemma 1: Consider composite-service templates that use only sequential and/or branch compositions, we have,

$$\forall y_l \in Y, \quad R(y_l) \geq \frac{n(y_l)}{C - g(y_l)}. \quad (37)$$

Proof: From equations (2)(11) and definition of $f(y_l, m_k)$ (see equation (35)), for composite-service templates that use only sequential and/or branch compositions, we have

$$R(y_l) = \frac{1}{t(y_l)} \sum_{m_k \in M} \frac{f(y_l, m_k)}{C - u_{m_k}}. \quad (38)$$

From the properties [15] of the convex function $u/(c - u)$, we have $\forall \theta_k > 0$ and $\sum_k \theta_k = 1$,

$$\sum_k \theta_k \frac{u_{m_k}}{C - u_{m_k}} \geq \frac{\theta_k u_{m_k}}{C - \theta_k u_{m_k}}. \quad (39)$$

Substitute θ_k with $\frac{f(y_l, m_k)/u_{m_k}}{\sum_{m_k \in M} f(y_l, m_k)/u_{m_k}}$,

$$\frac{1}{\sum_{m_k \in M} \frac{f(y_l, m_k)}{u_{m_k}}} \sum_{m_k \in M} \frac{f(y_l, m_k)}{C - u_{m_k}} \geq \frac{\sum_{m_k \in M} f(y_l, m_k)}{\sum_{m_k \in M} \frac{f(y_l, m_k)}{u_{m_k}} C - \sum_{m_k \in M} f(y_l, m_k)}, \quad (40)$$

we have

$$\sum_{m_k \in M} \frac{f(y_l, m_k)}{C - u_{m_k}} \geq \frac{f(y_l)}{C - g(y_l)} \quad (41)$$

Combined with equations (35)(39), we have

$$R(y_l) \geq \frac{n(y_l)}{C - g(y_l)}. \quad (42)$$

Lemma 2: Consider composite-service templates that only use sequential and/or branch compositions. We have

$$\forall y_l \in Y, \quad g(y_l) \leq C\check{\rho}(y_l). \quad (43)$$

Proof: We prove it by contraction. Assuming that $\exists y_l \in Y$, $g(y_l) > C\check{\rho}(y_l)$. Combine equations (26)(38), we get $R(y_l) > R(y_l)$. Contradiction. Therefore, we have proved Lemma 2.

Theorem 3 is proved as follows. We have

$$|M| = \sum_{y_l \in Y} \frac{f(y_l)}{g(y_l)} \geq \frac{1}{C} \sum_{y_l \in Y} \frac{f(y_l)}{\check{\rho}(y_l)} = \frac{\beta}{C} \sum_{y_l \in Y} \frac{\omega(y_l)n(y_l)}{\check{\rho}(y_l)}, \quad (44)$$

where the inequality is derived using result of Lemma 2, Therefore,

$$\beta \leq 1 / \sum_{y_l \in Y} \frac{\omega(y_l)n(y_l)}{C|M|\check{\rho}(y_l)}. \quad (45)$$

VIII. APPENDIX B

Proof of Theorem 7:

Recall that, in our algorithm, the deployment procedure itself is *optimal* for given decomposed response time requirement of atomic services. Therefore, it is sufficient to prove

$$\forall (s_i, y_l) \in A, \quad \check{\rho}^b(s_i, y_l) \leq \check{\rho}^h(s_i, y_l), \quad (46)$$

where $\check{\rho}^b(s_i, y_l)$ and $\check{\rho}^h(s_i, y_l)$ are the machine utilization thresholds under the baseline and our algorithm, respectively.

Let u be the utilization of all machines when throughput β is achieved under baseline algorithm. From equation (2), we have

$$r(v_{i,k}) = \frac{n_{s_i}}{C - u}. \quad (47)$$

The baseline algorithm does not have a response time decomposition step, but we can imagine a "pseudo-decomposition" step as follows. Let \tilde{n} and n_i be the CPU consumption of a high-level block and its i -th sub-block, and \tilde{r} and r_i be the response time of a high-level block and its i -th sub-block, respectively. We observe from equation (47) that, for parallel composition, the sub-block response time is equal to the high-level block response time scaled by the maximum CPU consumption of all parallel branches, i.e.,

$$r_i = \frac{n_i}{\max_i n_i} \tilde{r} \quad (\text{parallel composition}). \quad (48)$$

For sequential and branch compositions, the sub-block response time is proportional to its CPU consumption:

$$r_i = \frac{n_i}{\tilde{n}} \tilde{r} \quad (\text{sequential and branch composition}). \quad (49)$$

Comparing the decomposition methods in baseline algorithm (equations (48)(49)) with those in heuristic algorithm (equations (20)(21)), we see that the baseline algorithm and our heuristic algorithm have the same decomposition ratio (n_i/\tilde{n}) for sequential and branch composition blocks. However, for parallel composition, the baseline algorithm may generate lower response time for each sub-block than our algorithm does. Combining the above two arguments, we have proved equation (46), and thus completed the proof.