Received July 31, 2019, accepted August 29, 2019, date of publication September 9, 2019, date of current version September 20, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2939903

A Task-Resource Mapping Algorithm for Large-Scale Batch-Mode Computational Marine Hydrodynamics Codes on Containerized Private Cloud

YIYI XU[1,2], (Student Member, IEEE), PENGFEI LIU[3], IRENE PENESIS[1], AND GUANGHUA HE[4]

[1]Australia Maritime College, University of Tasmania, Launceston, TAS 7248, Australia
[2]School of Computer Science and Communication Engineering, Guangxi University of Science and Technology, Liuzhou 545006, China
[3]Marine, Offshore and Subsea Technology, School of Engineering, Newcastle University, Newcastle Upon Tyne NE1 7RU, U.K.
[4]Harbin Institute of Technology, Weihai 264209, China

Corresponding author: Pengfei Liu (pengfei.liu@gmail.com)

The work of Y. Xu was supported by the Australian Maritime College, University of Tasmania, by providing Ph.D. Scholarship.

ABSTRACT CPU time has long been a remaining problem for large-scale batch mode based scientific computing applications. To address this time-consuming problem, a container-based private cloud was employed, and a novel task-resource mapping algorithm was developed. Firstly, the execution features of typical batch mode codes were extracted and then computing jobs were formulated as a coarseness acyclic DAG. Secondly, to guarantee both job makespan and resource utilization, a novel task-resource mapping algorithm, along with container pre-planning and worst-case-first task placement phases, were developed. Finally, a typical Computational Marine Hydrodynamics software, Rotorysics, with a different scale of input data matrix was used as benchmark software. To manifest the effectiveness of the proposed method, a number of numerical examples were given via CloudSim and a small-medium containerized private cloud platform was adopted with three practical study cases. The computational results show that 1) compared with the traditional HPC workstation computing solution, container-based cloud solution shows significant savings in makespan by more than 6 times. 2) the new method is scalable to address bigger size batch computing problem up to a run matrix 10^8.

INDEX TERMS Computational marine hydrodynamics (CMH) codes, containerization, large-scale batch mode computing, private cloud, task-resource mapping algorithm.

I. INTRODUCTION

Computational Marine Hydrodynamics (CMH) is becoming more and more popular because of its cost-effectiveness and improved accuracy today. There are many commercial CMH packages available for a wide range of applications in marine industry. For example, CFD codes based on RANS methods, include such as CFX, Fluent, STAR-CD, and STAR-CCM+ (CD-Adapco), etc. These codes are capable of investigating local physics flow properties [1]. In addition, there are some much-less computing-intensive but many tasks in-house codes, especially developed for engineering design and optimization, such as PMARC [2], GASFLOW [3], Rotorysics (formerly Propella [4], [5], DF-OSFBEM [6]–[8]. These codes often use batch processing to run an executable repeatedly to obtain a large set of hydrodynamic performance data. Compared with RANS-based CFD codes, these computing tasks may require a relatively short CPU time, but many runs, in some cases, in an order of 10^5 or more.

For example, to design a turbine series prototype using Rotorysics software, a propeller and rotor design code, it took about 6 months of CPU time [9]. The computing job was completed by using multiple-threaded computation (48 executable running simultaneously) on a Dell Workstation (4-core 3.0 MHz CPU of 48GB of DRAM). The motion and geometry parameters are shown in Table 1 and Table 2 as an example. Along with the series of geometric and motion parameters, the number of tasks and the size of workloads

The associate editor coordinating the review of this manuscript and approving it for publication was Alberto Cano.

 This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

TABLE 1. Data matrix.

Variables	Parameters	Value
N_{rps}	rotor shaft rotational speed	10
N_{Vel}	number of inflow velocities	10
N_{EAR}	number of rotor solidity	9
N_{Pitch}	number of pitch /diameter ratio	9
N_z	number of blades	12
N_{Tasks}	$N_{rps} \times N_{Vel} \times N_{EAR} \times N_{Pitch} \times N_z$	105300

TABLE 2. Example of CPU run time.

Number of blades	Value
Nz=2	t=425 sec=0.12 hours
Nz=12	t=86400 sec=24 hours

TABLE 3. Current accelerated solutions.

Solutions	Benefits	Problem
Batch processing	• Works efficiency for repeated jobs. • Dominating method for long years.	• Heavily depends on HPC services and horizontal scaling . • Takes long CPU time to achieve required results.
Parallel processing	• Popular accelerated method with mature IDE	• NOT suitable for scientific codes with independent many-task with long delay added due to extra in-communication
Grid processing	• Performance insurance	• Complicated configuration and maintenance • Needs dedicated devices
Public cloud computing	• Pay on a demand-driven basis • Low maintenance cost	• Data security • NOT cost effective • Unstable performance if speed of network is low
Recode	• Methods and function updated	• Needs domain knowledge in-communication • High-level language means new challenges

can be calculated. It is obvious that to get more accuracy, more CPU time and memory are needed to complete such a relatively big data matrix run.

To improve computing efficiency for these large-scale scientific codes, many studies [10]–[16] were conducted ranging from software recode to parallel computing as shown in Table 3. Among them, cloud computing is an emerging technology and new trend to support scientific computing.

However, as scientific applications often rely on plenty of libraries, most of binaries and settled configuration files, have

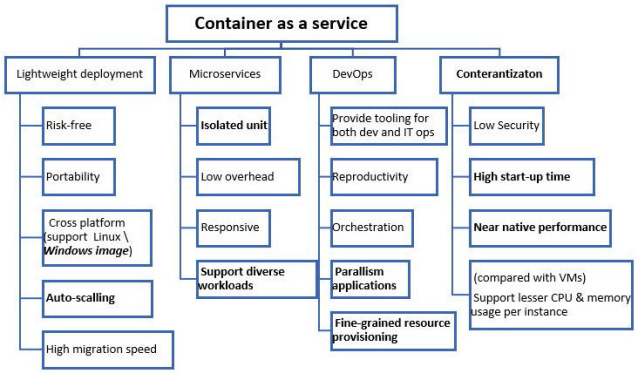


FIGURE 1. Advantages of container-based technical. Feathers in boldface are absorbed in our method.

series environment dependences [17], [18]. This means that migrating scientific codes from workstations on distributed platforms will encounter some compatibility problems, such as tool installation issues etc. In this regard, containerization technology is emerged to allow task scheduling on lightweight containers and support their migration whenever required. Compared with virtualization technology, containerization technology requires only a few seconds to bootstrap and initiate versus several minutes for a regular VM. Docker [19] and native workstation have no significant performance difference. As shown in Fig. 1, a new type of cloud service based on containerization, sontainer as a service (CoaaS), is considered efficient to address above issues.

For CoaaS, task scheduling algorithm is considered as a key factor to contribute to High Performance Computing (HPC), but it is still in its infancy [20]–[23]. With the development of container technology in industry, researchers put forward many optimal algorithms to enhance multiple goals, such as service cost, response time, and load balancing etc. It was also observed that these goals are covered from only service for cloud providers at system-level to users at user-level that aim to run expert system with high computing requirements in the last three years.

Xin *et al.* [24] adopted container-based virtualization technology and proposed a container-based scheduling strategy by using stable matching theory. Their strategy produced at most 27.4% degradation compared with virtual-machine-based clouds on response time. Their method proves to be good for small-medium scale computational tasks, especially the number of tasks increases to 200 or more.

Guan *et al.* [22] presented a dynamic allocation algorithm for Docker container resource named AODC which adapts to application's requirements. AODC algorithm takes into consideration the resources on nodes, the network consumption of nodes and the energy consumption of nodes. Simulation results indicate that AODC algorithm outperforms existing VM-based methods in terms of cost and custom acceptance ratio.

Luxiu *et al.* [25] developed a new task-scheduling and resource management algorithm for fog computing.

Their approach aimed to reduce task delays in accordance with the characteristics of containers. The results showed their method can effectively reduce task delays and improve the concurrency number of the tasks significantly.

Yanghu and Wenbin [26] conducted extensive experiments and results show that their algorithm can effectively reduce the system load imbalance and improve the overall system performance compared with other algorithms.

Tamanna *et al.* [27] proposed a genetic algorithm-based customer-conscious resource allocation and task scheduling. Their algorithm based on popular multi-cloud computing and consists of two phases. The first phase is a genetic algorithm-based resource allocation and the second phase is the shortest task first scheduling. These algorithms aimed to have minimum makespan and maximum customer satisfaction. Results from simulation illustrate that the proposed algorithm outrun the existing ones as per concerned metrics.

Jose *et al.* [23] provided a resource allocation solution by extending the concept of time slicing to the level of the container. By using this approach, they can control and mitigate some of the more detrimental performance effects over subscription. Their results show significant improvement over standard scheduling of Docker.

Singh *et al.* [28] designed a renewable energy-based host selection and container consolidation scheme. The proposed approach has been evaluated using Google workload tracers. The results obtained are 15%, 28% and 10.55% higher energy savings in comparison to the existing solutions.

Chanwit and Kornrathak [29] proposed a container scheduling algorithm with Docker Swarmkit. In their study, they presented an Ant Colony Optimization (ACO) strategy to improve the scheduler's optimality and introduce these specific parameters to make the scheduler works better for each situation. Finally, their experimental results show ACO strategy is better than greedy algorithm by approximately 15% on resource usages. Their purpose is to balance the use of resources so that applications in a container cluster will have a better performance.

Zulkar *et al.* [30] evaluated a dynamic fuzzy load balancing algorithm to predict the virtual machine where the next job will be scheduled. This is quite similar to the idea we propose in Section 3, where we focus on the resource allocation for light-weighted containers instead of virtual machines.

Pande *et al.* [31] proposed a customer-oriented task scheduling algorithm for a heterogeneous multi-cloud environment. The basic idea of this algorithm is to assign a suitable task for each cloud which takes minimum execution time. It balances the makespan by inserting as many as tasks into idle slots of each cloud, so that time can be minimized. A simulated result proves the validity of this method.

Li *et al.* [32] proposed PINE—a performance isolation optimization solution in container environments when different services (latency-sensitive services and throughput first services) are deployed on a host machine using container technology. Their experimental results show that PINE can effectively optimize the performance of running services and achieve the optimal result in a relatively short time.

Kwon *et al.* [33] firstly proposed several hurdles to realize container-enabled clusters for HPDA (high-performance data analytics) workload; Secondly, to address technical constraints around bottlenecked inter-connections for overlay networking and storage access, they designed a container-enabled cluster prototype. Their results show an effectively and seamlessly integration with the hardware and software pieces of HPDA cluster.

Zhuang *et al.* [34] proposed an immediate access for Earth science modeling. They stated that cloud is a promising vehicle for massively-parallel simulations emulating local HPC clusters, but several issues need to be addressed. Software containers hence used to support GEOS-Chem global model of atmospheric chemistry and they gave a detailed description of how its software environment to be moved smoothly between cloud platforms and local clusters by using container technical. Their work provides general guidance for scientific models to cloud computing platforms in a user accessible way.

Lin *et al.* state that there are some open issues have not been completely addressed in the deployment and management of the microservice container. In their work, a multi-objective optimization model is built, and an ant colony algorithm is proposed to solve the scheduling problem. By comparing with other related algorithms, the experimental results show their ant colony algorithm is effective for container resource scheduling not only satisfies the service requirements of users but also reduces the running overhead.

In addition, it is interesting that Waibel *et al.* [35] presented a container-based eBPMS, named ViePEP-C, for the process activity execution. Their work is somewhat similar to our work by using a novel resource and task scheduling algorithm in containers-based execution environment. The main difference is their work is designed for business processing system, but our work focused on scientific workflow. Their experiment results shown that the execution cost can be decreased by over 20% by comparing with other state-of-the-art VM-based scheduling algorithm.

Traditional cloud scheduling policy focused on ''let resources adapt computational tasks''; recent containerization technology make it possible to ''let computational jobs adapt resources'', by well-scheduling jobs to reserved containers as mentioned above. However, most of related studies rely on public cloud infrastructure and hence brings some extra service expenses. For non-expert cloud end-users, dynamic configuration management of a container and the relationship with containers and nodes are also a source of confusion. In our work, an containerized private cloud is exploited with an automatic container deploy technology, and corresponding novel job-resource mapping algorithms are developed. Finally, according to the above steps, makespan time needed for implementation is reduced, and computing performance is enhanced significantly.

The remainder of this paper is organized as follows. Section 2 introduces our methodology, such as detailed module design and key strategy. Section 3 presents evaluation and results of the proposed methodology by using a simulator and a real private cloud test bed. Section 4 concludes the work and suggests some future work.

II. METHODOLOGY

To achieve the above goals, a framework with 3 modules is proposed as shown in Fig. 2. These 3 modules are functioned as job formalization and classification, reservation of container and job-container scheduler.

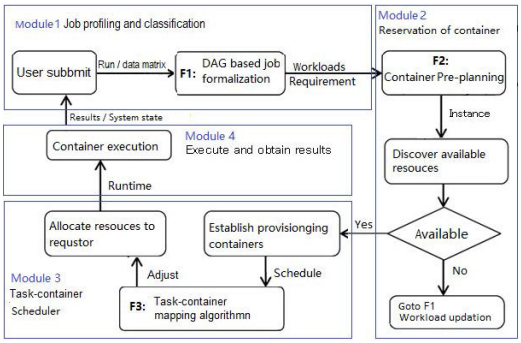


FIGURE 2. Flow chart of the proposed methodology.

Given the characteristics of tasks, Module 1 is a pre-process to describe how users will gain access and interact with cloud platform. This access is implemented by a DAG based quantization job queueing and several constrains with requested demand. Module 2 is a pre-process to estimate and predict instances of containers, or in other words, maximum resource consumption for each task. The instance here consists of configuration, numbers (maximum amount) and layers (we assume that instances of containers can be nested if needed). This module hence can provide global information at scheduling time for next step. Module 3 aims to schedule jobs in containers under a minimum makespan and a maximum utilization. As these two objectives have conflicts, in this work, we developed an advanced worst-case-first bin packing algorithm to address the time-consuming question. Module 4 is a nature execution process on parallel containers and end users can obtain computing results for further analysis.

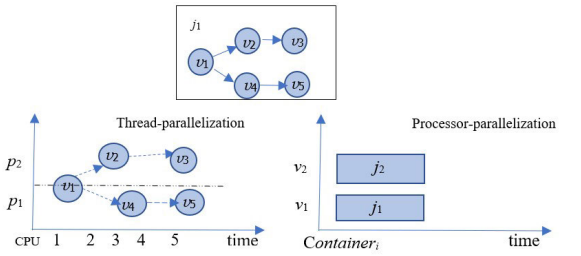
A. JOB FORMALIZATION AND CLASSIFICATION

Large-scale batch mode computing jobs based CMH applications often requires a relatively short CPU time to obtain one data point but need a huge number of runs to build performance database as mentioned above. To characterize this kind of jobs, the notations is given in Table 4, where,

1) Each J is separated into n operations, and each operation is named "task" here. Each job J consists of multiple tasks, which is in a range from 10^5 to 10^{12} or even more.
2) Resources and container are considered the same.

TABLE 4. Table of notations.

Notation	Meaning
W	cloud workflow service set
J	a workflow ensemble (job sets)
H	length of schedule (scheduling horizon)
j_i	job i
v_i	container i
p_i	processor i
S_{j_i}	start time of j_i
E_{j_i}	end time of j_i
T_{j_i}	execution time of j_i
Re_{v_j}	capability of container of v_j
C_{v_j}	CPU usage of the container v_j
P_{v_j}	processors sets which belongs to the containerv_j
Re_{j_i}	acquired resources of job j_i



a. Execution in one folder on Workstation b. Execution in one container on Cloud

FIGURE 3. Execution features.

3) Each J is non-pre-emptive tasks. This means that if two tasks belong to the same job J, they must be mapped on the same processor in the same container as shown in Fig. 3.
4) Each J is indivisible with unit execution time, and hence it has less DAG defined time and inter communication time.
5) To weighted total overhead CPU run time of the process W, one generally needs to considerate DAG defined time, average memory access latency, task waiting time and communication time, and makespan time etc. It is noted not every time metric can be quantified or determined easily. Some factors have only marginal effects according the features of batch-mode workflows and private run platform. Therefore, in our work makespan time is used as the main performance index to be optimized and evaluated.
6) According the fluctuation and orders of magnitude of workloads, each J belongs to one of three different workflows. For convenience in the succeeding discussions, we name these workflows as W_k, where $k = 1 \vee normal, 2 \vee uniform, 3 \vee fluctuation$.
7) Every given type of W_k has the same stochastic run process and hence the minimum and maximum numbers of required containers can be predicated.

Several definitions are introduced as following.

Generally, $W = \{DG, T, Dl, C\}$ used to describe cloud workflow sets, where, DG represents DAG rule for workflows ensembles W, T is the total CPU run time of the process W, $Dl = \{Dl_j \mid j \in [1, m]\}$, is a lease lifetime for containers, and C is the service fee for W per unit time.

It is noted that a large body of research modelled as DAGs as: $G = \{V, E, S\}$, where $V = \{v_1, v_2, v_3, \ldots, v_m\}$ is the task node set of DAG graph, v_i is the *ith* task, n represents the number of tasks; E is a set of edges, where, $E = \{e_{ij}\}$, $i, j \in Q$, e_{ij} represents the precedence constraint relations such that j_j cannot complete its execution before j_i begins. S is the set of servers, $S = \{S_1, S_2, \ldots, S_k\}$, S_k represents the *kth* server, k represents the number of servers.

In this work, we assume one job, or a scientific workflow that ensembles as an integrated unit. Therefore,

Definition 1: $G = \{J, E, V, Map\}$, where, J represents a set of jobs in W, j_i means *ith* job of this set; $J = \{J_1, J_2, J_3, \ldots, J_n\}$, where n is the number of jobs.

$V = \{v_1, v_2, \ldots, v_m\}$ represents the sets of containers, where v_i presents *ith* container, m is the number of containers (machines). $E = \{E_j \mid j \in [1, H]\}$ represents job dependencies in W, where H is the length of schedule (scheduling horizon). In addition, dependency between j_i and j_j can be stated as, j_i is precursor node of j_j, and j_j is successor node of j_i. $Map = \{(J, v_m) \mid v_m \in V_m\}$, expounded the mapping relationship between job J and container v_m, Map_{J-v_m} means that the job J is assigned for v_m.

Definition 2: $v_j = \{U_{v_j}, Re_{v_j}, C_{v_j}, Pe_{v_j}\}$ is attribute sets of one container, where U_{v_j}, represents the resource usage of a job j_i for a container v_j; Re_{v_j} represents the capability of container v_j, $P_{v_j} = \{P_{v_{j_e}} \mid e \in [1, n]\}$ represents n processors which belongs to the same container v_j, and C_{v_j} is the cost of container v_j per unit time.

Definition 3: $j_i = \{S_{j_i}, E_{j_i}, Re_{j_i}, Ear_{j_i}, Tard_{j_i}\}$ denotes attribute sets of one job, where S_{j_i}, E_{j_i}, T_{j_i}, standing for start time, end time and execution time of the job, where $E_{j_i} = S_{j_i} + T_{j_i}$, stands for a sub-makespan of J. The $Re_{j_i} = \{re_{j_q^i}\} \mid_{q=1}^{3}\}$ represents the type, size and configuration of resources that job j_i acquired, include CPU, memory and storage.

If $T_{j_i} < E_{j_i}$, earliness occurs and if $T_{j_i} > E_{j_i}$, tardiness occurs, Ear_{j_i} and $Tard_{j_i}$ will be used to represent earliness of j_i and tardiness of j_i in scheduling process, respectively.

$$f(Deviation) = \frac{1}{T}\int_T \frac{Re_{v_j} - Re_{j_i}}{Re_{v_j}} dt, [-1, 1] \tag{1}$$

Basic performance matrix are defined as following:

Definition 4: $makespan = max(< E_{j_i} >| j_i \in J)$. *makespan* represents the total time needed to execute an entire J, that is, equal to the end time of the last task to be completed.

Definition 5: $cost = (\sum_{j=1}^{m}(C_{v_j} \times t(j, V_{v_j}), S)$, *cost* represents the service price of J per unit time for server S.

Definition 6:

$utilization(J, V_c) = 1 - \frac{\sum_{f=1}^{m} \times u_{v_j} - \sum_{f=1}^{n} \times Re_{v_j}}{\sum_{f=1}^{m} \times u_{v_j}}$ represents the resource utility of job J, where, n is the number of jobs, and m the number of containers, while Map_{J-v_m}.

The problem then can be formulated as: Given a weighted DAG $G = \{J, E, V, Map\}$ and m containers, the optimal problem consists of minimize the makespan, cost and maximization utilization of J, as described in equation (2):

$$f(Requirement) = \begin{cases} min(makespan) \le Dl_i (i \in [1, m]) \\ min(cost) \le Cv_i (i \in [1, m]) \\ maximize(utilization(J, V_c)) \end{cases} \tag{2}$$

In the following, the constraints of scheduling problem are described.

$$\left.\begin{array}{l} \sum_{k=1}^{H} kE_{j_i} \le E_J \\ \sum_{k=1}^{H} kS_{j_i} \le S_J \\ \sum_{k=1}^{H} kT_{j_i} \le T_J \end{array}\right\}, j_i \in J; i = 1, 2, \ldots, n; \quad k = 1, 2, \ldots, H-1 \tag{3}$$

Equation (4) ensures that each J is separated into n operations, which have different unit execution time.

$$\sum_{k=1}^{n} T_{j_i} \le 1, \quad k = 1, 2, \ldots, H-1 \tag{4}$$

Equation 4 ensures that in each time slot, most one part of job can be performed or the container may be idle at the time slot.

B. CONTAINER PRE-PLANNING STRATEGY

In this section, we introduce a container pre-planning strategy. This strategy obtains flexibility instances for current jobs according to the specific context. In a cloud environment, there are three usual provision strategies. They are on-demand, reserved and spot resource [36]. For on-demand strategy, it is a popular one which is charged in a short-term in a pay as-you-go manner, but performance is unpredictable due to a possible external load. The reserved strategy is a simple and native one to provide fixed instances (resource capacities of CPU, memory and disk for selection) and that does not change over time. Reserved instances require a high upfront investment, but have 2 to 3 times lower per-hour cost than on demand resources. For spot strategy, we ignore it as they do not provide any availability guarantees. To offer better availability and consistent resource for batch mode jobs introduced above, a hybrid resource provision mechanization is adopted. This process can be considered as on demand provision at the initial step.

There are three steps in this pre-process. Firstly, according a set of required resources Re_{j_i}, for each $j_i \in J$, convert the workload histogram to maximum likelihood distribution fitting by using maximum likelihood distribution fitting function [37]. While workload type and corresponding instances

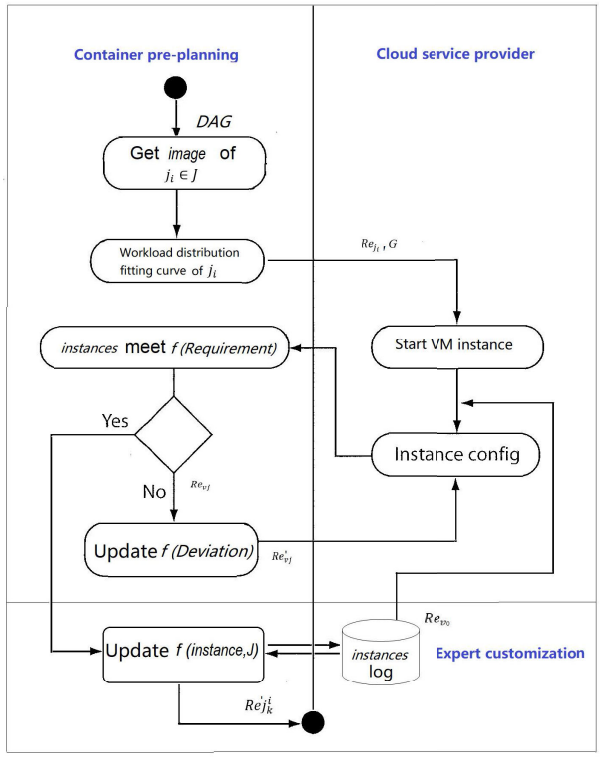


FIGURE 4. Hybrid container pre-planning provision model.

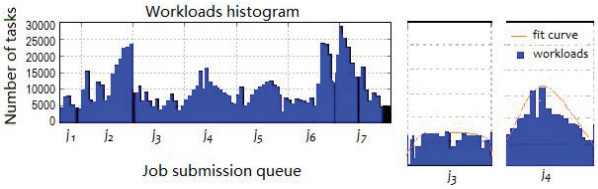


FIGURE 5. Part of workloads histogram by using Weinman's model.

be identified, the resource provision is a static process for the whole job J. A specific process is shown in Fig. 4. It is noted that the part of the idea comes from Genady *et al.* [38].

Secondly, to ensure asymptomatic accuracy of selecting appropriate instance size and configuration, this provision model adopted Weinman's measurement model [39], [40] as a deviation function. This model illustrates how much resource is needed for the jobs to work properly. Fig. 5 illustrates examples, the curves show a hypothetical situation with a normal distribution variation in demand (solid blue line), linearly increasing resource provision (dotted black line), and white perpendicular lines represents idle time. Weinman's measure is a weighted combination of the areas between curves. According this conceptual model, we improved the deviation function in this work as in equation (5).

$$f(Deviation) = \frac{1}{T}\left(\int_T \frac{Re_{v_j} - Re_{j_i}}{Re_{v_j}} dt\right), [-1, 1] \qquad (5)$$

where, Re_{v_j} and Re_{j_i} are the provisioned and the demanded resources respectively at time t. T is the time period in which

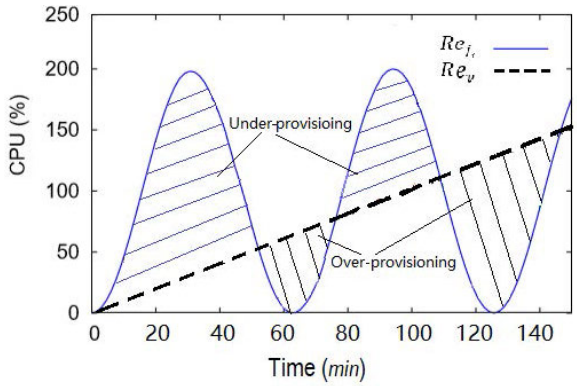


FIGURE 6. Geometric view of *f* (*deviation*) for *normal* workloads.

we measure the two metrics. The integral value measures the relative squashed area enclosed by the resource values of Re_{j_i} and Re_{v_j} for time period T. The value of deviation ranges between $[-1, 1]$, indicating an under-provisioning situation when $f(Deviation) \in [-1, 0)$; and an over-provisioning case when $ff(Deviation) \in (0, 1]$. Fig. 6 shows the geometric view of deviation result. The zero value indicates an accurate provision of resources according to the demanded resources. The bigger the result of $f(Deviation)$ close to 1, a larger correction is needed between Re_{v_j} and Re_{j_i} for Cloud service.

For each $J \in G$, $f(Instance, job)$ is a function used to identify and save optimal instance and job sets. By using above optimal distribution fitting diagram, when most of the utilization of containers can reach 90% or more, the detail capacity of instances will be locked and saved. Each instance here including three index values of resource, which corresponding the peak, average and minimum Re_{v_j} of current W, respectively. Finally, all instances scheme for past scenarios saved to historical log. While workloads of new scenarios are close to one of *Js* in this historical log, corresponding the set of pair $< instance, job >$ will be recommended directly.

Algorithm 1 shows the pseudo-code of the container pre-planning algorithm.

C. CONTAINER-BASED TASK-RESOURCE MAPPING ALGORITHM

In this section, we introduces an advanced scheduling policy to fully exploit containers. As mentioned, Docker has some simple policies with low time complexity [29], [41]. Amongst these polices, the core idea is to deploy jobs to machines with higher CPU usage and higher memory usage and bin packing algorithms, such as First Fit, Best Fit, Best Fit Decreasing, etc., have been proposed to solve this problem [42]. The disadvantage of this policy is that it naturally spreads the workload over the entire cluster of machines, which in turn leads to overload some nodes easily [43]. Based on a bin packing algorithmic enhancement, we set up a worst-case first bin packing job-resource mapping algorithm and assume worse-case often has maximum makespan and bottleneck in scheduling lines. This algorithm parallelizes each

Algorithm 1 Container Pre-Planning Algorithm

Input: $W = \{DG, T, Dl, C\}$;
$J = J_1, J_2, \ldots, J_n$;
$J_i = \{S_{j_i}, E_{j_i}, T_{j_i}, Re_{j_i}; Ear_{j_i}, Tard_{j_i}\}$;
$v_j = \{U_{v_j}, Re_{v_j}, C_{v_j}, Pe_{v_j}\}$;
$V_c = \{\emptyset\}$.
Output: find Optimal Re_{v_j} for V_c,
obtain instances sets: $< instances, J >$.

Begin
(1)For each $j_i \in J$, convert the workload histogram to maximum likelihood distribution fitting and search maximize resource requirement Re_{j_i};
(2) If exist $< instance, J >$, then update V_c with instances;
(3) If $Re_{j_i} \leq Re_{j_i}$ and $f(requirement) = True$;
// whether the container v_j can meet the resource requirement of job j_i.
$Re_{v_j} \leftarrow Re_{j_i}$, Update Re_{v_j},$V_c \leftarrow V_c \cup V_j$; move J_i from J;
(4) Else $deviation = f(Deviation)$;
(5) If $deviation \leq 0$, Re_{v_j} is over-provision, decrease level of instance;
(6) else $v_{m_e}++$; Re_{v_j} is under-provision, increase the level of instance;
(7) End If
(8) End If
(9) Update Re'_{j_i}, $Re'_{v_j} \leftarrow Re'_{j_i}$; *Goto* (2)
(10) If $average(C_{v_i}) \geq 90\%$, and $J = \{\emptyset\}$, $instances = instances \cup f(instance, J)$;
(11) update *historical log*;
(12) End If
(13) $j_i = j_i++$; *Goto* (1)
(14) End For
End

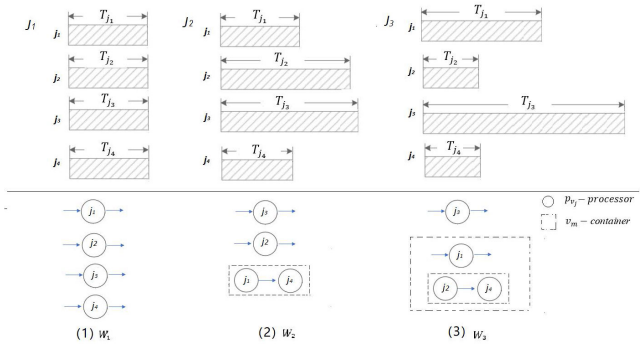


FIGURE 7. The process of parallel job merging.

job according to their maximum finished time and available resources. The basic idea is illustrated in Fig. 7.

As shown in Fig. 7, in the first phase, jobs are descending according to T_{j_i}. The worst-case with maximum execution time was moved as the first job and "guide one" for other jobs. In the second phase, it computes the execution time T_{j_i} of other jobs. If T_{j_i} do not exceed the "guide one" and the idle time also is minimum according to Equation (3), these tasks will be merged with next one until it cannot be merged. In the final phase, each job will be assigned to processors which has the earliest S_{j_i} in a container.

Detailed flow and pseudo-code of this strategy is discussed in *Algorithm* 2.

Algorithm 2 Container-Based Task-Resource Mapping Algorithm

Input: $W = \{DG, T, Dl, C\}$;
$J = J_1, J_2, \ldots, J_n$;
$J_i = \{S_{j_i}, E_{j_i}, T_{j_i}, Re_{j_i}, Ear_{j_i}, Tard_{j_i}\}$;
$v_j = \{U_{v_j}, Re_{v_j}, C_{v_j}, Pe_{v_j}\}$;
$V = \{v_1, v_2, \ldots, v_m\}$.
Output: Re-schedule G and assigned each J to certain container and processors.

Begin
(1) For each W_k;
(2) If number of $Ear_{j_i} \approx$ number of $Tard_{j_i}$, $k = normal \vee 1$; if number of $Ear_{j_i} \geq$ number of $Tard_{j_i}$, $k = uniform \vee 2$; else $k = fluctuation \vee 3$;
(3) End If
(4) End For
(5) For each $V_i \in V$, descending sorted according to their maximum resource usage u_{v_i}; Update $V = V'$;
(6) End For
(7) For each $j_i \in J$, descending sorted according to their maximum execution time T_{j_i}. Update $J = J'$;
(8) End For
(9) For each $J \in DG$
(10) While $k == 1$;
(11) If $T_{j_i} \leq makespan(J')$ and $V_c \neq \{\emptyset\}$;
(12) $Re'_{v_j} = min\{Re_{j_i}\}$;
(13) $Map_{j_i - v_j}$; $i \leftarrow i++$;
(14) End If
(15) Update Re'_{j_i}, $Re'_{v_j} = Re'_{j_i}$; *Goto* (7)
(16) If $J = \{\emptyset\}$, $instances = instances \cup f(instance, J)$;
(17) Remove j_i and update J'_i; *Goto* (7)
(18) End If
(19) While $k == 2$ or $k == 3$;
(20) $j_i \leftarrow maximum\{T_{j_i}\}$;
(21) If $Ej_i \leq T_{j_i}$ and $V_c \neq \{\emptyset\}$;
(22) $Re'_{v_j} = min\{Re_{j_i}\}$;
(23) $Map_{j_i - v_j}$;
(24) Else $j_i \leftarrow j_i \cup j_(i++)$; recalculate $E_{j_i} + (E_{j_i}++)$;
(25) End If
(26) If $E_{j_i} + (E_{j_i}++) \leq Dl$;
(27) Update $makespan(J') = T_{j_i}$;
(28) $Re'_{v_j} = min\{Re_{j_i}\}$;
(29) $j_i \leftarrow j_{i++}$;
(30) $Map_{j_i - v_j}$; $i \leftarrow i + 1$;
(31) Remove j_i; Update J'_i;
(32) Else $j_i \leftarrow j_i \cup j_{i++}$; recalculate $E_{j_i} + E_{j_i++}$; *Goto* (25);
(33) End If
(34)$Map_{J - v_m}$; Updated G; *// The job is mapped to resource.*
(35) End For
End

III. RESULTS AND DISCUSSION

Given the inherent dynamical environment of a private cloud computing environment and the prohibitive costs, it is difficult to conduct physical performance evaluation in a repeatable, scalable and controllable manner [11]. Even when we restrict the application to the same implementations of the same algorithm, substantial variations in performance will still exist on the same distributed computers. Therefore, we used a simulation package CloudSim based tool [44] to evaluate the effectiveness of the proposed framework and algorithms. To be able to present clearly, three terms will further explain here. "Workload" is the whole amount of work (or jobs) an MTC CMH running to ending. Hence, workload can be classified as type, size etc. As mentioned above, we named it W_1, W_2, W_3 as three typical workloads. "Scenario" describes the different run environment for workloads, such as isolated workstation, private cloud, container-based cloud etc. We named it P1, P2, P3 in our experiments. "Case" used in our paper relates to real world engineering applications and more details listed in Table 7. We then compared with the baseline bin packing algorithm, Docker adopted currently, under three different scenarios. Next, to compare computational performance and accelerated impact, we conducted three typical case studies on both physical container-based private cloud and traditional workstations. Simulation and experiment results are presented and analysed as follows.

TABLE 5. Parameters setting of cloudsim in Private cloud.

Entities	Parameters	Value
Containers	• Type ♯1 • Type ♯2 • Type ♯3 • Storage	• CPU MIPS 4685; Memory 128 • CPU MIPS 9320; Memory 256 • CPU MIPS 18636; Memory 512 • 500GB
Data centre	• Number of data centre	• 1 (100 hosts)
Job	• Total number of tasks • Length of task	• $10^3; 10^5; 10^8$ • 5000 - 15000MI

A. TESTING IN SIMULATION

1) IMPLEMENTATION SETTING

The algorithm was programmed in ContainerCloudSim 4.0 in Windows server 2016. In addition, we assume memory and disk capability are unlimited in a private cloud environment, and only considerate CPU in our simulation environment. Table 5 shows main simulation parameters. Network bandwidth is assumed to be 1 GBps and 400 KBps for servers and containers.

By using ContainerCloudSim, for simulating scientific computing job sets, we simulated three workload patterns, *uniform*, *normal* and *random* as mentioned. For specific provision and placement policy, we extended our algorithms by overriding the functionalities with some classes of CloudSim. Major related classes we used are shown in Table 6. For each workload scenario, its ideal duration has no scheduling delays or node failures.

TABLE 6. Basic class used in containercloudsim.

Name	Function
Workload Management	support variable workloads
CloudletScheduler	share processor
CloudInformationService	share of processing power among Cloudlets
ContainerAllocationPolicy	allocate containers to VMs
VmAllocationPolicy	support optimize allocation method

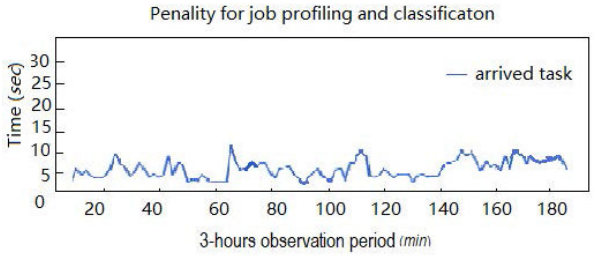


FIGURE 8. An example of processing time of module 1.

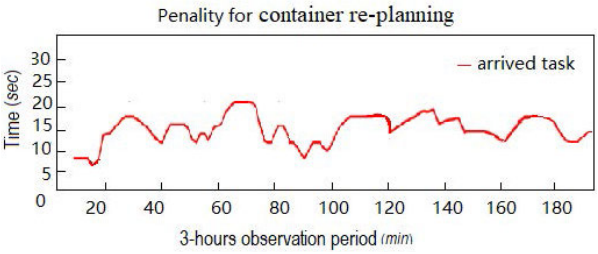


FIGURE 9. An example of processing time of module 2.

2) ALGORITHM EFFECTIVENESS

Overheads: In the presented strategies, the mapping algorithm includes two pre-treatment process modules as shown in Fig. 2. They are Module 1, job profiling and classification, Module 2, container re-planning with on-demand instances. In Fig. 8 and 9, the pre-processing time related with container is shown, while considering the normal distribution scenario with a typical 10^5 run matrix in 3 hours.

Fig. 8 shows that job profiling and classification takes 5 to 10 seconds on average and only happens at the first time after a job is submitted. Fig. 9 shows that container re-planning does not exceed 20 seconds. Totally, it induces lower than 0.1% overheads to the execution time of jobs by comparing without using our strategy. Therefore, container-based cloud works well in our research with similar performance to nature machine and hence extra penalty of time consuming can be ignored.

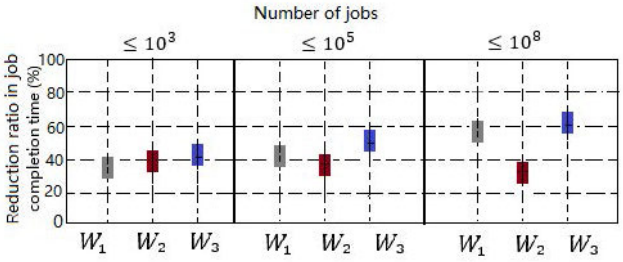


FIGURE 10. Reduction ratio with different type and size of workloads.

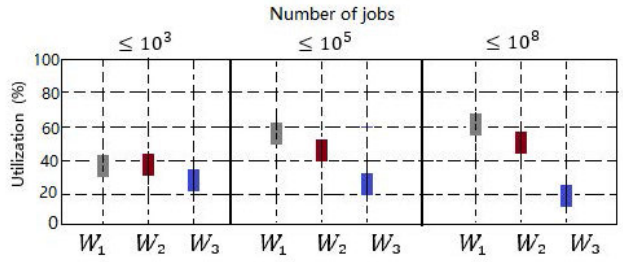


FIGURE 11. Utilization ratio with different type and size of workloads.

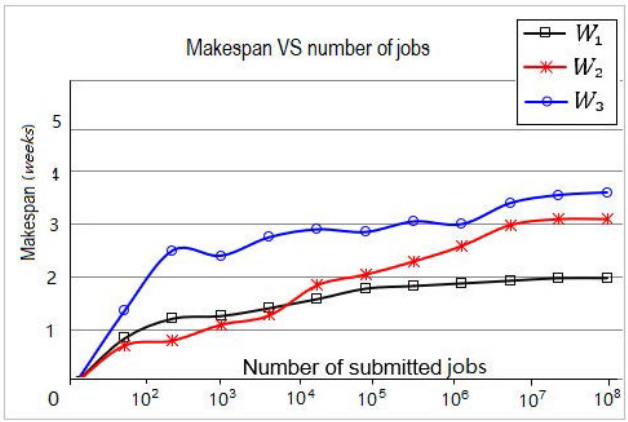


FIGURE 12. Makespan Vs number of tasks.

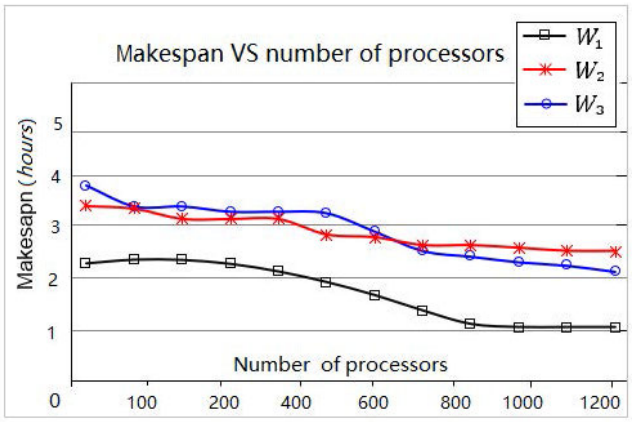


FIGURE 13. Makespan Vs number of processors in 5-hours periods.

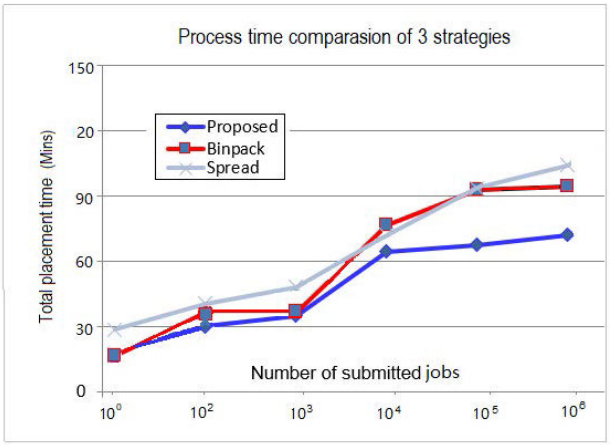


FIGURE 14. Makespan Vs number of tasks.

Adaptability: It is clear from Fig. 10 and 11 that all types of workloads W_1,W_2,W_3, benefit significantly from the proposed strategy in both CPU run time and utilization of resources by using our strategy. These results show under the heavy batch mode loads, the strategy also exhibits a good performance. Specifically, Fig. 10 shows that proposed strategy is most effective for W_3 and W_1 workloads. While the number of jobs increased significantly, the time-reduction ratio trend declined around 10 percent or so.

Fig. 11 shows the utilization ratio by using our strategy. The strategy is most effective for W_1 and W_2 workloads, and this trend is stable along with the increased run matrix. Compared with W_1 and W_2, resource utilization of W_3 achieves a reduction of 17% to 30%.

Fig. 12 and 13 show the specific impact of average makespan for these scenarios W_1, W_2,W_3 under a different size of jobs and a different number of processors, respectively. As expected in container pre-planning strategy, Fig. 12 demonstrates the effectiveness of the proposed algorithm even run matrix up to 10^8. Fig. 12 also shows that our proposed approach work best for the type of W_3. In Fig. 13, we observed the change of makespan in 5-hours periods for the same J while we increased the number of processors. The result shows that the proposed strategy can be converged to an ideal state to find the maximum size of instances. In other words, for batch mode computing, computational performance is not always increased when the number of processors is increased.

Total placement time comparison: The developed advanced worst-case-first bin packing algorithm is compared with other two original task placement algorithms, they are spread, and bin packing algorithm, respectively. As can be seen in Fig. 14 and 15, the total execution time of the proposed algorithm is better than the other existing task placement algorithms. It is noted when running job achieve to 10^3, it achieves an optimal point for Binpack. However, the current algorithm increases the processor utilization by 7% to 10% while the number of running jobs is increased to 10^5 or more as shown in Fig. 14.

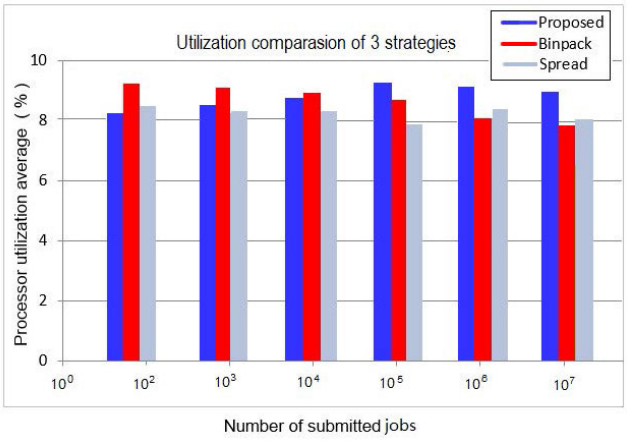


FIGURE 15. Makespan Vs number of processors.

Also, we found Amdahl's law [45], [46] is applicable for container-based Cloud computing, that meaning processor utilization is not always increasing with the number of jobs. For the two traditional mechanisms with many containers, it is possible being useful only for highly parallelized programs but not good for MTC codes. The implied reason could be that non-parallelization part of the MTC codes determines whole processing time. For proposed algorithm, it is also can find a threshold. As shown in Fig.15, when running jobs achieves to 10^6, processor utilization presented a decreasing trend. However, the Binpack strategy packs containers into machines as few as possible, that it is too easy to overload some nodes; The Spread strategy places tasks evenly based on the specified value and hence it is inelastic. These are reasons why they can not maximize the resource reuse while compared with proposed algorithm. In our proposed techniques, as introduced in section 2 we developed a hybrid allocation with container pre-planning strategy to determine required resources according context. This method not only consider benefits of cloud provider but also customers. The proposed algorithm has the ability to scale the number of processors along with increased tasks, which provides higher returns in resource utilization as we can seen in Fig.15.

B. EXPERIMENTAL RESULTS

The sensitivity of previous findings in simulation to marine engineering practice was evaluated. In the experiment three marine engineering cases with real-world datasets were set up by using software Rotorysics as benchmark application as shown on 127953. The basic purpose is to test the effect of changes in the size of input parameters and type of workflow. CPU time of each job can be tracked easily as left part of in Fig. 1. In addition, detailed configuration of cloud environment is listed in Table 8. As containers are OS-specific, to run windows image, Docker is chosen as container. We repeated the experiments three times in a mini private cloud (one data centre and configuration similar to Table 5) and an isolated DELL workstation (6-cores CPU, 128GB memory and 1TB HDD), respectively. It is noted that our mini cloud

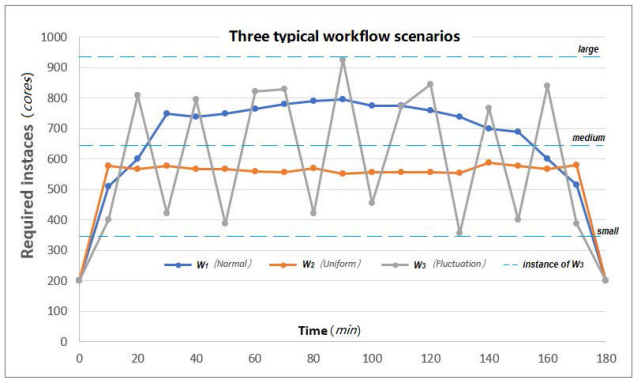


FIGURE 16. Example of the process stream of bath-mode application.

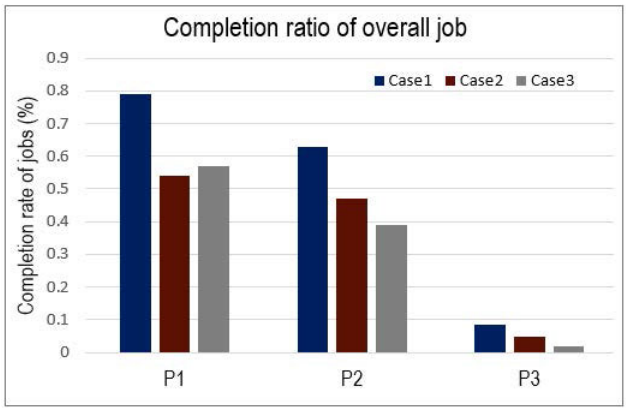


FIGURE 17. Completion rate of overall job in a 5-hour period.

infrastructure closely resembles a private IaaS cloud that one would find easily in small to medium scale enterprise environments.

Fig. 16 illustrates the results of *algorithm*1, Model 2. For the three practical cases, the required resources for the job can be assigned with certainty. Instance family is identified as large, medium and small ones.

As can be seen in Fig. 16, for the *uniform* scenario, the aggregated resource requirements are 59 cores. The difference between the maximum and the minimum load is 10% and each individual job is 0.9 hours shorter. For normal scenarios, the minimum load requires an average of 47 cores, while the maximum load requires 89 cores and most job lasts 1 to 2 hours. For a fluctuation scenario, the minimum load drops at 36 cores, while the maximum load requires 97 cores. On an average 55% jobs it needs 4 hours but other jobs less than 1 hours. The experiments show that our strategy is a high customized resource allocation and hence with a high CPU utilization.

Next, suppose P1, P2, P3 to represent 3 experimental platforms, they are container-based cloud with the new proposed algorithm, container-based cloud without new algorithm and traditional workstation with batch mode solution, respectively. All study cases 1, 2 and 3 where to be performed in P1, P2 and P3 three times. The comparison of computing performance of these platforms are shown in Fig. 17 and 18.

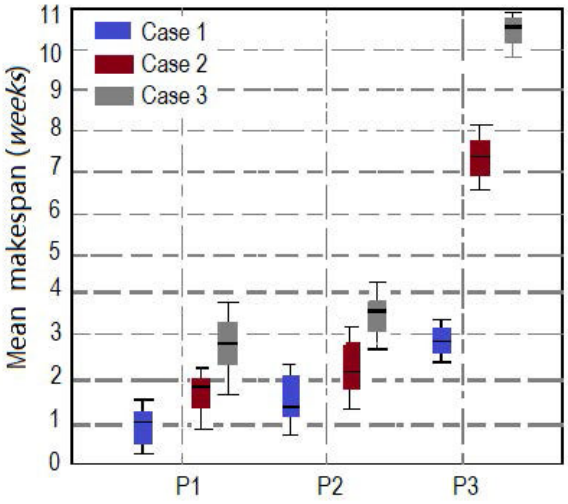


FIGURE 18. Mean makespan for 3 cases in 3 platforms.

TABLE 7. Application cases.

Case No.	Workloads	Distribution characteristics	Application goals
Case 1	medium taskset with 10^3	normal	New series of turbine rotors
Case 2	medium taskset with 10^5	uniform	Liu and Bose (2012)
Case 3	nmedium taskset with 10^8	fluctuation	Optimization design of Wing-In-Ground

TABLE 8. Environment parameters setting.

Entities	Parameters
Operating system	Windows server 2016
Test applications	Rotorysics
Cloud development platform	Microsoft Azure 2017
Container cluster systems	Docker Swarm 2017 (modified)
Docker monitor	DocSnap or DocLite
Container orchestration	Apache Marathon
Benchmark	Apache Benchmark

As can be seen in Fig. 17 and Table 8, in terms of the 5 initial hours observed period, assuming on the same workload and same case, processing in P1 and P2 are far more than P3 in completion rate as expected. Fig. 18 shows the mean makespan with variance in P1, P2, P3; the mean CPU run time saving of P1 is 6 and 3 times of P3 and P2, respectively; The results are similar to the simulation results shown in Fig. 12. These results are further proved that our approach was efficacious in computational performance.

Finally, customer satisfaction rate, which is generally used as a comprehensive market index [47]. In our work, it is calculated from 3 parameters which feedback from monitor, including overall time to completion, waiting time for resource and resource utilization. By comparing with $f(Requirement)$, the final comparison can be seen in Fig. 19. There are large margins between P1, P2 and P3: while the amount of input data increases to 10^5, the satisfaction rate of P1 and P2 is 35% and 41% higher than that of P3, respectively; while the amount of input data increases to 10^8, the satisfaction rate of P1 and P2 is 27%,47% respectively, higher than P3.

TABLE 9. Mean makespan of different cases.

Case	P1	P2	P3
Case1	0.25	1.18	2.9
Case2	1.1	2.3	7.4
Case3	2.15	3.6	10.8
Mean makespan	1.166667	2.36	7.033333

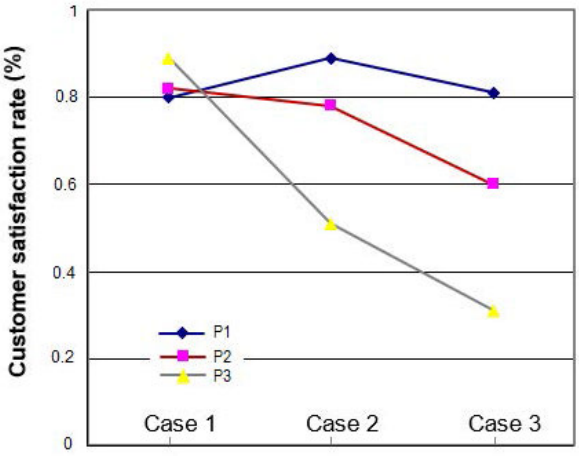


FIGURE 19. Customer satisfaction rate vs. Input data matrix.

IV. CONCLUSION

In this paper, we address a new and important problem concerning the long CPU time consuming issue for CMH codes under limited cloud resources over containerized private clouds. The developed scheduling algorithm propose a new task-resource match stragety to facilitate the collaboration between end-users and service providers efficiently.

The simulation results show that our approach delivers improvements over baseline algorithms at fast scheduling level, resource utilization and overall performance. The experimental results show that compared with research on traditional workstation, container-based private cloud could be used as an accessible lightweight high-performance computing platform to accelerate their research circle significantly. The proposed approach presents an enhancement of 6 times, 44% in achieved makespan reduction and average custom satisfaction rate, respectively.

To the authors' knowledge, this work is the first systematic research and development study to use predictable instances and limited paralleled containerized resource to reduce CPU execution time for CMH codes. This proposed algorithms can be applied to other large-scale batch processing situations in other disciplines.

The limitations of our methodology were also observed. Container used in our work is exclusive in a whole job execution time. It is clear that conflict exists and affects other business services in the private cloud. This effect is not currently measured in the work. However, as the mean utilization of CPU of data centre was low and the technique to exploit edge computing capacity of user nodes was developed these years, the related problem can be further solved.

ACKNOWLEDGMENT

The Harbin Institute of Technology, Weihai, and the Guangxi University of Science and Technology, China, are provided support to the Y. Xu with Cloud facilities and technical assistance.

REFERENCES

[1] K. A. Hoffmann and S. T. Chiang, *Computational Fluid Dynamics* (Engineering Education System), vol. 1. Wichita, KS, USA: Engineering Education System, 2000.
[2] D. Ashby, M. Dudley, S. Iguchi, L. Browne, and J. Katz, *Potential Flow Theory and Operation Guide for the Panel Code PMARC* (Engineering Education System). Moffett Field, CA, USA: NASA Ames Research Centre, 1991.
[3] J. Xiao, J. R. Travis, W. Breitung, and T. Jordan, "Numerical analysis of hydrogen risk mitigation measures for support of ITER licensing," *Fusion Eng. Des.*, vol. 85, no. 2, pp. 205–214, 2010.
[4] P. Liu, "Software development on propeller geometry input processing and panel method predictions of propulsive performance of the R-class propeller," MMC Eng. Res., Newfoundland, BC, Canada, Tech. Rep. 1, 1996.
[5] P. Liu and N. Bose, "An unsteady panel method for hihgly skewed propellers in non-uniform inflow," in *Proc. Propeller RANS/Panel Method Workshop (ITTC)*, Grenoble, France, B. Gindroz, T. Hoshino, and J. Pyllkanen, Eds., Apr. 1998, pp. 343–350.
[6] P. Liu, "A time-domain panel method for oscillating propulsors with both chordwise and spanwise flexibility," Ph.D. dissertation, Memorial Univ. Newfoundland, Newfoundland, BC, Canada, 1996.
[7] P. Liu and N. Bose, "Propulsive performance from oscillating propulsors with spanwise flexibility," *Proc. Roy. Soc. London A, Math., Phys. Eng. Sci.*, vol. 453, no. 1963, pp. 1763–1770, 1997.
[8] P. Liu, "A computational hydrodynamics method for horizontal axis turbine—Panel method modeling migration from propulsion to turbine energy," *Energy*, vol. 35, pp. 2843–2851, Jul. 2010.
[9] P. Liu and N. Bose, "Prototyping a series of bi-directional horizontal axis tidal turbines for optimum energy conversion," *Appl. Energy*, vol. 99, pp. 50–66, Nov. 2012.
[10] K. A. Iskra, R. G. Belleman, G. D. van Albada, J. Santoso, P. M. A. Sloot, H. E. Bal, H. J. W. Spoelder, and M. Bubak, "The polder computing environment: A system for interactive distributed simulation," *Concurrency Comput., Pract. Exper.*, vol. 14, nos. 13–15, pp. 1313–1335, 2002.
[11] R. Buyya, S. K. Garg, and R. N. Calheiros, "SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions," in *Proc. Int. Conf. Cloud Service Comput.*, Dec. 2011, pp. 1–10.
[12] S. Sakr, A. Liu, D. M. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 3, pp. 311–336, 3rd Quart., 2011.
[13] G. Mateescu, W. Gentzsch, and C. J. Ribbens, "Hybrid computing—Where HPC meets grid and cloud computing," *Future Gener. Comput. Syst.*, vol. 27, no. 5, pp. 440–453, May 2011.
[14] V. Kindratenko and P. Trancoso, "Trends in high-performance computing," *Comput. Sci. Eng.*, vol. 13, no. 3, pp. 92–95, 2011.
[15] A. W. Z. Chew, T. Vu, and A. W.-K. Law, "High performance computational hydrodynamic simulations: UPC parallel architecture as a future alternative," in *Proc. Int. Conf. Comput. Sci.*, Jun. 2018, pp. 444–455.
[16] P. Liu, N. Bose, K. Chen, and Y. Xu, "Development and optimization of dual-mode propellers for renewable energy," *Renew. Energy*, vol. 119, pp. 566–576, Apr. 2018.
[17] K. Liu, K. Aida, S. Yokoyama, and Y. Masatani, "Flexible container-based computing platform on cloud for scientific workflows," in *Proc. Int. Conf. Cloud Comput. Res. Innov. (ICCCRI)*, May 2016, pp. 56–63.
[18] S. Yokoyama, Y. Masatani, T. Ohta, O. Ogasawara, N. Yoshioka, K. Liu, and K. Aida, "Reproducible scientific computing environment with overlay cloud architecture," in *Proc. IEEE 9th Int. Conf. Cloud Comput.*, Jun./Jul. 2016, pp. 774–781.
[19] Z. Kozhirbayev and R. O. Sinnott, "A performance comparison of container-based technologies for the cloud," *Future Gener. Comput. Syst.*, vol. 68, pp. 175–182, Mar. 2017.
[20] D. Zhang, B.-H. Yan, Z. Feng, C. Zhang, and Y.-X. Wang, "Container oriented job scheduling using linear programming model," in *Proc. 3rd IEEE Int. Conf. Inf. Manage.*, Apr. 2017, pp. 174–180.
[21] B. Varghese, L. T. Subba, L. Thai, and A. Barker, "Container-based cloud virtual machine benchmarking," in *Proc. IEEE Int. Conf. Cloud Eng.*, Apr. 2016, pp. 192–201.
[22] X. Guan, X. Wan, B.-Y. Choi, S. Song, and J. Zhu, "Application oriented dynamic resource allocation for data centers using docker containers," *IEEE Commun. Lett.*, vol. 21, no. 3, pp. 504–507, Mar. 2017.
[23] J. Monsalve, A. Landwehr, and M. Taufer, "Dynamic CPU resource allocation in containerized cloud environments," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2015, pp. 535–536.
[24] X. Xu, H. Yu, and X. Pei, "A novel resource scheduling approach in container based clouds," in *Proc. IEEE 17th Int. Conf. Comput. Sci. Eng.*, Dec. 2014, pp. 257–264.
[25] L. Yin, J. Luo, and H. Luo, "Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4712–4721, Oct. 2018.
[26] Y. Guo and W. Yao, "A container scheduling strategy based on neighborhood division in micro service," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Apr. 2018, pp. 1–6.
[27] T. Jena and J. R. Mohanty, "GA-based customer-conscious resource allocation and task scheduling in multi-cloud computing," *Arabian J. Sci. Eng.*, vol. 43, no. 8, pp. 4115–4130, 2018.
[28] N. Kumar, G. S. Aujla, S. Garg, K. Kaur, R. Ranjan, and S. K. Garg, "Renewable energy-based multi-indexed job classification and container management scheme for sustainability of cloud data centers," *IEEE Trans. Ind. Informat.*, vol. 15, no. 5, pp. 2947–2957, May 2019.
[29] C. Kaewkasi and K. Chuenmuneewong, "Improvement of container scheduling for docker using ant colony optimization," in *Proc. 9th Int. Conf. Knowl. Smart Technol. (KST)*, Feb. 2017, pp. 254–259.
[30] M. S. Q. Zulkar, M. A. K. Azad, S. Abdullah, and R. M. Rahman, "Fuzzy logic based dynamic load balancing in virtualized data centers," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Jul. 2013, pp. 1–7.
[31] S. K. Pande, S. K. Pande, and S. Das, "A customer-oriented task scheduling for heterogeneous multi-cloud environment," *Int. J. Cloud Appl. Comput.*, vol. 6, no. 4, pp. 1–17, 2016.
[32] Y. Li, J. Zhang, C. Jiang, J. Wan, and Z. Ren, "PINE: Optimizing performance isolation in container environments," *IEEE Access*, vol. 7, pp. 30410–30422, 2019.
[33] J. Kwon, N. L. Kim, M. Kang, and J. WonKim, "Design and prototyping of container-enabled cluster for high performance data analytics," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2019, pp. 436–438.
[34] J. Zhuang, D. J. Jacob, J. F. Gaya, R. M. Yantosca, E. W. Lundgren, M. P. Sulprizio, and S. D. Eastham, "Enabling immediate access to earth science models through cloud computing: Application to the GEOS-Chem model," *Bull. Amer. Meteorolog. Soc.*, to be published.
[35] P. Waibel, C. Hochreiner, S. Schulte, A. Koschmider, and J. Mendling, "ViePEP-C: A container-based elastic process platform," *IEEE Trans. Cloud Comput.*, to be published.
[36] C. Delimitrou and C. Kozyrakis, "HCloud: Resource-efficient provisioning in shared cloud systems," *ACM SIGARCH Comput. Archit.*, vol. 44, no. 2, pp. 473–488, 2016.
[37] D. Cousineau, S. Brown, and A. Heathcote, "Fitting distributions using maximum likelihood: Methods and packages," *Behav. Res. Methods, Instrum., Comput.*, vol. 36, no. 4, pp. 742–756, 2004.
[38] G. Grabarnik, M. Klems, L. Shwartz, S. Tai, and C. Ward, "System and method for reducing latency time with cloud services," U.S. Patent 9 098 456 B2, Aug. 4, 2015.

[39] J. Weinman, "Time is money: The value of 'on-demand,'" *Joe Weinman. Com*, p. 30, Jan. 2011.
[40] S. Islam, K. Lee, A. Fekete, and A. Liu, "How a consumer can measure elasticity for cloud platforms," in *Proc. 3rd ACM/SPEC Int. Conf. Perform. Eng.*, 2012, pp. 85–96.
[41] C. Boettiger, "An introduction to docker for reproducible research," *SIGOPS Oper. Syst. Rev.*, vol. 49, no. 1, pp. 71–79, Jan. 2015.
[42] A. Caprara, "Properties of some ILP formulations of a class of partitioning problems," *Discrete Appl. Math.*, vol. 87, nos. 1–3, pp. 11–23, 1998.
[43] J. O. Iglesias, M. De Cauwer, D. Mehta, B. O'Sullivan, and L. Murphy, "Increasing task consolidation efficiency by using more accurate resource estimations," *Future Gener. Comput. Syst.*, vol. 56, pp. 407–420, Mar. 2016.
[44] R. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011.
[45] L. Yavits, A. Morad, and R. Ginosar, "The effect of communication and synchronization on Amdahl's law in multicore systems," *Parallel Comput.*, vol. 40, no. 1, pp. 1–16, 2014.
[46] J. Nutaro and B. Zeigler, "How to apply Amdahl's law to multithreaded multicore processors," *J. Parallel Distrib. Comput.*, vol. 107, pp. 1–2, Sep. 2017.
[47] J. Mei, K. Li, and K. Li, "Customer-satisfaction-aware optimal multiserver configuration for profit maximization in cloud computing," *IEEE Trans. Sustain. Comput.*, vol. 2, no. 1, pp. 17–29, Jan./Mar. 2017.



YIYI XU received the B.A. degree from the Hunan University of Science and Technology, China, in 2003, and the master's degree in computer engineering from the Huazhong University of Science and Technology, China, in 2006. She is currently pursuing the Ph.D. degree with Australian Maritime College, University of Tasmania. Prior to her study, she was an Associate Professor with the Guangxi University of Science and Technology (GXUST), China. She has taught computer network-related courses for more than ten years at GXUST. Her current research interests include interdisciplinary field and computational marine hydrodynamics.



PENGFEI LIU received the B.Eng. degree from the Wuhan University of Technology (WUT), China, in 1982, and the M.Eng. and Ph.D. degrees in naval architecture from the Memorial University of Newfoundland (MUN), Canada, in 1991, and 1996, respectively. He was a Senior Research Officer with National Research Council Canada, from 1999 to 2016. He has been an Associate Professor with the Australian Maritime College, University of Tasmania (UTAS), Australia, since 2016. He has been an Adjunct Professor/Researcher with MUN, since 2000; China Ocean University, from 2002 to 2005; Institute of Mechanics, Chinese Academy of Sciences, from 2005 to 2008; Harbin Ship Engineering University, China, from 2008 to 2010; UTAS, from 2013 to 2016; and Harbin Institute of Technology, Weihai, since 2017. He is currently a Professor of hydrodynamics with the Marine, Offshore and Subsea Technology, School of Engineering, Newcastle University, Newcastle Upon Tyne, U.K. He has involved intensively for over two decades in the development of specialty engineering software, teaching undergraduate, and supervision of higher degrees by research. He is a Professional Engineer of APEGBC, Canada, and a member of various international academic committees, including the Board of Directors of the Computational Fluid Dynamics Society of Canada, from 2001 to 2007, and the ISSC Ocean Space Utilization Committee, from 2018 to 2021. He has coauthored over 120 refereed journal articles and conference articles in engineering software development for hydrodynamic/aerodynamic applications of rotary and oscillatory wings for propulsion and renewable energy.



IRENE PENESIS received the Ph.D. degree in mathematics from RMIT University, Melbourne, VIC, Australia, in 2002. Her Ph.D. was in the field of tribology examined the pressure field and load-carrying capacity specialized in non-smooth gas-lubricated bearings used in industrial drilling machinery. She developed mathematical models to solve the complex elliptic partial differential equations analytically and numerically that governed the pressure field. Before joining the Australian Maritime College (a special institute of the University of Tasmania), she was a Lecturer with the School of Mathematical and Geospatial Sciences, RMIT University, and served a short postdoctoral research role with the Department of Mathematics and Statistics, University of Melbourne, modeling shape changes of red blood cells. She is currently an Associate Professor and a Leader of the Marine Renewable Energy Research Group, University of Tasmania. She is also a Bid Director of Blue Economy CRC. She was a member of the Royal Institution for Naval Architects (RINA).



GUANGHUA HE received the B.Eng. and M.Eng. degrees from Tianjin University, and the Ph.D. degree in atmospheric and marine environmental engineering from Osaka University, Osaka, Japan, in 2014. He is currently a Professor and a Ph.D. Supervisor with the Harbin Institute of Technology (HIT), Weihai, China. He was a member of The International Society of Offshore and Polar Engineers (ISOPE).