A Scheduling Framework for Periodic Tasks in Geo-Distributed Data Centers

Yan Li[1,2,3], Hong Zhang[3], Yong Wang[3], Xinran Liu[3], Peng Zhang[4]
[1]University of Chinese Academy of Sciences, Beijing 100080, China
[2]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[3]National Computer Network Emergency Response Technical Team/Coordination Center, Beijing 100029, China
[4]Institute of Information engineering, Chinese Academy of Sciences, Beijing 100093, China
E-mail: liyan@ncic.ac.cn, zhangh@cert.org.cn, wangyong@cert.org.cn, lxr@isc.org.cn, pengzhang@iie.ac.cn

***Abstract*— we present *FPT*, a scheduling framework for periodic tasks that uses the temporal characteristic of periodic tasks to alleviate the overhead of geo-distributed data centers. In *FPT*, clients are able to express the periodicity about their tasks, and this characteristic is used to realize one-time scheduling for multiple executions. For a set of periodic tasks, aiming to find the minimum number of VMs required to guarantee the scheduability and generate the task execution sequence on each VM, an algorithm is also presented. A case-study and its evaluation are given to show the efficiency of our framework.**

key words*: *task scheduling; periodic task; geo-distributed data centers; cloud computing

I. INTRODUCTION

With the prevalence of cloud computing services, there has been a growing trend toward geographically distributed data centers. Google has built dozens of data centers all over the world to guarantee the quality of internet service to global users [1]. One of the greatest challenges in leveraging these data centers is efficient task scheduling. Better energy efficiency, geographical load balancing and fairness are extensively discussed in previous works [2, 3, 4, 5, and 6].

However, cloud task scheduling is an NP-hard optimization problem. Due to existence of different workload types with various requirements that should be supported by data centers, no any single schedule strategy can allocate resources to all imaginable types efficiently. For example, numerical computing tasks are usually CPU intensive, while database operations typically require high-memory support. The heterogeneity of workload demands poses significant technical challenges on the schedule mechanism, giving rise to many delicate issues -notably efficiency -that must be carefully addressed [7].

Existing approaches to workload characterization for cloud computing mainly focus on task resource requirements for CPU, memory, disk, I/O, network, etc. However, in addition to resource requirements, tasks frequently have placement constraints or temporal characteristics of the execution. Table 1 show data taken from our production geo-distributed data centers called iVCE test bed over a period of 31 days in this July. On average, there are 18 million tasks need to schedule per day. Most of these tasks are very short and the average running time is 3 minutes. Notably, more than half of them are periodic task. Actually the test bed especially the scheduler is facing significant pressure to allocate resources efficiently. In some extremes, the success ratio of task scheduling is less than 50%. After analysis, the low success ratio is mainly attributed to the high load. As shown in table 1, there are more than 200 tasks need to schedule average per second. The test bed has used a centralized management approach, in which a super master node schedules tasks among the Geo-Distributed data centers. In another side, there exist many complicated task placement constraints such as on OS versions, machine types, physical place and network accessing methods, those lead to high computational-complexity to find an optimal resource to execute a task. Using benchmarks of Google compute clusters, the results of experiment in [8] indicate that the presence of constraints increases task scheduling delays by a factor of 2 to 6, which often means tens of minutes of additional task wait time.

TABLE 1. MEAN VALUE IN JULY

Source	Number of tasks	Average running time	Number of periodic tasks
iVCE test bed	**18 million**	**180 seconds**	**10.5 million**

In general, distributed scheduling scheme can reduce the pressure. But this is beyond the scope of this paper. We try to use the temporal features- notably periodicity -to optimize the scheduling process.

Despite the unprecedented heterogeneity in geo-distributed data centers, state-of-the-art computing frameworks have paid little attention to the temporal features of workload.

However, the periodic features increase the difficulties of efficiently scheduling at least from the following two aspects:

(1) The heavy overhead on the master node resulting from the highly frequent scheduling. As shown in table 1, there are more than 200 tasks need to schedule average per second.

(2) The pressure of too much bandwidth consumption. Though Geo-Distributed data centers are usually connected together with dedicated high-bandwidth communication links, the bandwidth is limited during the peak-hour. Assuming that the average size of tasks is 1 MB, it will consume 200MB/S bandwidth to distribute tasks.

978-1-4799-8356-8/15 $31.00 © 2015 IEEE
DOI 10.1109/SOSE.2015.29

In this work, we focus on the method to alleviate the pressure of the scheduler which is the key component in Geo-Distributed data centers. The number of tasks those need to be scheduled in unit time is the most essential factor for determining the pressure of the scheduler. Reducing the number of scheduling times is most simple and intuitionistic. Periodic tasks can be scheduled one-time for repeated execution at regular intervals.

This paper presents a novel framework called *FPT*, which aims to reduce the unnecessary scheduling for periodic tasks in Geo-Distributed data centers.

The paper is organized as follows: Section II briefly introduce the Geo-Distributed data centers, and task models we consider. Then we show *FPT* design details in Section III. Section IV presents a scheduling solution under *FPT* framework for periodic tasks. A case-study and its evaluation are given in Section V. We survey the related work in Section VI. Section VII concludes the paper and future work.

II. SYSTEM AND TASK MODELS

This section describes the Geo-Distributed data centers system model and our task model.

A. The Geo-Distributed Data Centers System

In this paper, we consider an Platform-as-a-Service (PaaS) system, in which a number of Geo-Distributed data centers participate in a federated approach. These data centers provide basic on-demand storage, compute and network access capacities. The provision of these computational resources is in the form of virtual machines (VMs) deployed in data centers. Virtual machine is an abstract unit of storage, compute and network access capacities provided in a data center. VMs from different data centers are offered in different types, each of which has different characteristics. For example, they may have different numbers of CPUs, amounts of memory, network access points and network bandwidths.

We use a centralized management approach, in which a super node schedules tasks among multiple data centers. For each task submitted by clients, the super node decides which data center will execute this task based on the information from the task description.

In our model, there are N geographically distributed data centers GDC_i, $\forall$i ($0 \leqslant i \leqslant N-1$ unless stated otherwise). The physical location of GDC_i is denoted by pl_i. Furthermore, GDC_i has V_i VMs. Each VM contributes m resources (e.g., CPU, memory, storage and network) denoted by VM = {*cpu, mem, str, …, net*}, which means the VM can offer the capacity of *cpu* CPUs, *mem* GB memory and *str* GB storage. Network is a key resource in our model, which needs to be described at least in three dimensions, such as network operators (Enumerated. e.g., China Mobile -*CM* for short, China Unicom –*CU* for short, China Telecom –*CT* for short), the way of network access (Enumerated. e.g., ADSL, Optical Access Network abbreviated as OAN) and the bandwidth (Numeric, the values is in megabytes per second). Network resource can be denoted by net = {*no, na, bw*} accordingly.

B. Task Model

In this paper, we consider a single set of independent tasks $\Gamma = \{t_0, t_1..., t_{n-1}\}$, task t_i is defined using the following parameters (t_i is a tuple $\{id_i, pl_i, rq_i, st_i, ext_i, pr_i, ft_i\}$).

- id_i: The ID of t_i;
- pl_i: The physical place that t_i needs to execute in;
- rq_i: The resources those t_i requires, e.g., CPU, memory, storage and network described in section A;
- st_i: The start time for t_i. By default, st_i is equal to the release time;
- ext_i: The computation time requirement;
- pr_i: The period, means t_i needs to execute in every pr_i minutes($pr_i \geqslant ext_i$). In case of non-periodic task, pr_i is set to zero;
- ft_i: *The finish time*. It can be a relative value of st_i (e.g., 1 month). In case of non-periodic task, ft_i is set to zero;

For periodic task, t_i needs to execute once in every pr_i minutes between st_i and ft_i. The k^{th} execution of t_i is denoted by t_{ik}. Further, all periodic tasks are ready for execution at the beginning of each period.

III. THE SCHEDULING FRAMEWORK FOR PERIODIC TASKS

This section presents the system design of the scheduling Framework for Periodic Tasks (*FPT*). First, we describe a motivational example to identify the additional load incurred by scheduling periodic tasks instances over and over again. Then, we introduce our solutions to alleviate the overhead for scheduling. And last, a high level *FPT* architecture is presented.

A. Motivational Example

We give an example of periodic task in our Geo-Distributed data centers called iVCE test bed as mentioned in previous section .The iVCE test bed has been providing services for more than 10 applications. One of the most typical applications is China Internet Speed Test (*CIST*).

However, it is a challenging job to test the internet speed, especially in China. Firstly, there are several network operators including China Mobile, China Unicom and China Telecom. The capability of their network infrastructure is different. Secondly, there exists spatial imbalance of the Internet development in China, resulting in coexist of several generations' network access technologies. Finally, the cost is very high especially for large-scale test. Fortunately, the iVCE test bed provides platform services, which greatly reduces the challenge.

For the Website of high traffic burden, the internet speed to access a single page will become a bottleneck of the system. Therefore, the internet speed test for specific web sites (C*IST4SWS*) is meaningful, which is a very practical case in *CIST*. Because the network access speed changes with time, in general, *CIST4SWS* has to execute at a certain

period (10 minutes or much shorter). As shown in Fig. 1, there is a sequence of tasks aiming at testing the internet speed to access a target website over China Mobile network in Tibet and Beijing. These tasks can be divided into two sets according to resource requirements. Tasks within the same set are almost identical besides their execution time (execute once every 10minutes). However, all the tasks need to be scheduled before their execution in the mainstream of known cloud computing framework [9, 10, 11, 12, and 13]. Complicated task placement constraints lead to high computational-complexity to find an optimal machine to execute a task, which may increase task scheduling delays by a factor of 2 to 6 [8]. We need to optimize the scheduling process to alleviate the overhead on the scheduler.

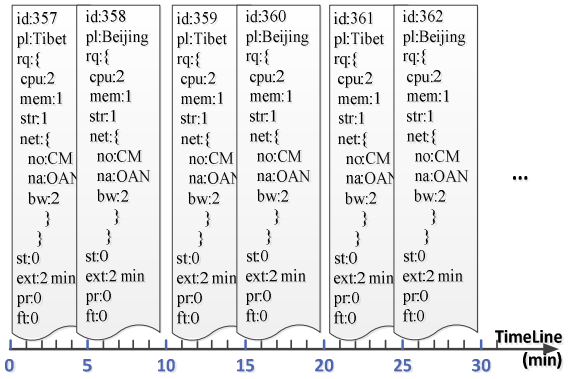

Fig. 1. Periodic Execution of tasks for *CIST4SWS*

B. Design Philosophy

In Geo-Distributed data centers, scheduling and monitoring already incur heavy overhead on the master (scheduler) node, which can easily become the bottleneck. In addition, frequent task release easily result in too much bandwidth consumption. Given this consideration, several design decisions have to be made to alleviate the overhead on the scheduler and communication links.

There are two expensive operations at the time of starting a task: scheduling and task distribution. How to reduce the number of these expensive operations while satisfying user's QOS requirements is a main challenge.

If we know a sequence of tasks are the instances of a periodic task, one-time scheduling for multiple executions can realize. In essence, periodicity is decided by the characteristics of upper applications. The users (or data center service consumer) can indicate the periodicity about a task clearly using our model mentioned in section II.

C. FPT Architecture

The architecture of *FPT* is illustrated in Fig.2. *FPT* is mainly composed of four modules:

(1) Information Collector (IC). Resources allocations are made according to the status messages about virtual machine across data centers. For example, how many resources are available for scheduling. Due to the highly dynamic information about the resources, IC must collect the information in time to facilitate task scheduling. There are two methods to collect the information, pull model and push model. In order to guarantee the collection effect, the hybrid method is used occasionally;

(2) Model Analyzer (MA). MA is responsible for parsing task descriptions, analyzing the resources requirements and grouping the tasks based on the requirement. The grouping can facilitate task scheduling;

(3) Execution Sequence Generator (ESG).In order to allocate resources among different tasks efficiently, ESG adopts special strategy to use virtual machines as few as possible to execute tasks in the same group. First, ESG acts as the match-maker between virtual machines and tasks. Then, ESG try to find minimum number of virtual machines from the candidate set for each task group, that can be seen as a special case of the bin packing problem. Lastly, the execution sequence about tasks in every selected virtual machine is generated. We will illustrate the process in detail in section IV;

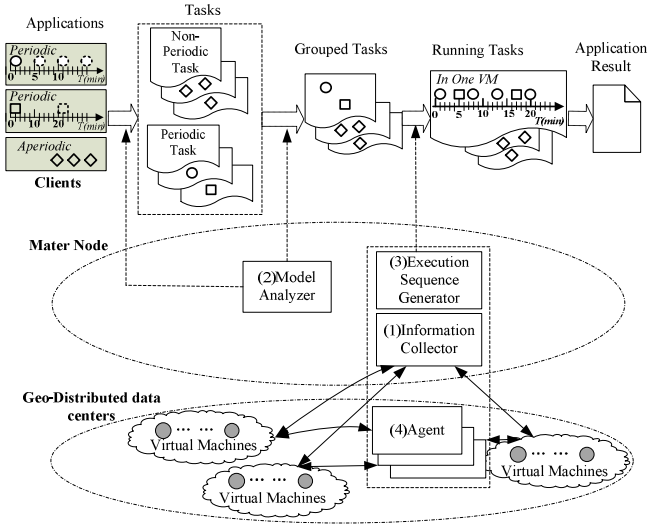

Fig. 2. FPT Architecture

(4) Agent. A single agent runs on each virtual machine, mainly focuses on two-fold roles. The first is to collect local information and status periodically, and report them to IC for further scheduling judgment. The second role is to ensure task processes to execute in predefined sequence.

To eliminate the interference between tasks which can cause incorrect test results, tasks in the same virtual machine are executed serially. Agent will start processes for a task only if it's time for execution. Specifically, when the execution time of a task is greater than the claimed value and the time for another task is up, processes belong to the former task will be killed to maintain execution order. For periodic tasks, the failure instance at a few points has little influence on the test results of the whole application. Therefore, we have not used the fault tolerance mechanism here.

IV. TASK EXECUTION SEQUENCE GENERATION ALGORITHM

ESG is the core functional module in *FPT*. How to find minimum number of virtual machines (VM) from the candidate set for each task group and generate task execution sequence in every selected resource is the main problem in ESG. With the models introduced in section II, the problem can be formally formulated as follows.

Problem 1 Given a task set $\Gamma = \{t_0, t_1..., t_{n-1}\}$ and VM set $V = \{vm_0, vm_1..., vm_{m-1}\}$ as described above, where each task is characterized by its computation time ext_i and its period pr_i, what is the minimum number of VMs required to execute the tasks in Γ such that all the tasks in the same VM are schedulable, and how to generate the task execution sequence for each selected VM.

We borrow existing solutions in Real-time operating system [14]. In recent years, much research has focused on the multiprocessor scheduling of real-time tasks problem [15, 16, 17, 18, and 19]. In work [15], a scheduling problem called Rate-Monotonic Multiprocessor Scheduling (or *RMMS*) is studied, and a new heuristic algorithm called RM-First-Fit-Decreasing-Utilization (or *RM-FFDU*) is shown for RMMS problem. RMMS is a bin-packing problem and has been proven to be NP-complete. To solve the RMMS problem, two issues need to be addressed: the scheduling on each processor and the assignment of tasks to processors. A sufficient schedulability condition for the scheduling of tasks on a single processor is given in [15]. *RM-FFDU* uses the famous First-Fit-Decreasing heuristic to assign tasks to processors.

Our problem is similar with the RMMS problem. We present an algorithm based on *RM-FFDU* to address the problem 1, the pseudo code as follows:

Algorithm 1.
Input: task set Γ, VM set V;
Output: m, linked list- l_1 to l_m.

(1) Sort Γ in the order of non-increasing utilization (for each task t, utilization u = $t.ext$ / $t.pr$);
(2) i=1; m=1;
linklist l_1;
(3) j=1;
While($u_i > 2(\prod_{l=1}^{k_j}(u_{j,l}+1))^{-1}-1$)
j=j+1;
//task t_i cannot be scheduled in VM_j
(4) $k_j = k_j + 1$; // k_j is the number of tasks assigned on VM_j.
(5) if (j>m)
{ m=j; linklist l_j;}
l_j.addinOrder(t_i.id);//keep in order of ascending pr
//Assign task t_i to VM_j,
(6) i=i+1;
(7) if (i>n) Exit;//n is the number of tasks in Γ
Else Goto (3).

When the algorithm returns, the value of m is the number of VMs needed to schedule task set Γ, l_j is the set of tasks assigned on VM_j, k_j is the size of l_j. Tasks in l_j are ordered by scheduling priority descending. The scheduling of tasks on VM_j can be done using the rate monotonic algorithm [20].

TABLE 2. TASKS BE SCHEDULED

order	Task id	ext (min)	pr (min)	U=ext/pr
1	331	2	5	0.4
2	320	1	3	0.333
3	321	1	5	0.2
4	280	2	10	0.2
5	281	1	10	0.1

Table 2 shows an example to illustrate the process. There are five tasks need to be scheduled. The sum of their utilization value is greater than 0.7435 ($5*(2^{1/5}-1)$) [20]. These tasks are not schedulable in a single VM. We have to partition the task set. Frist, task 331 is assigned on VM_1. For task 320, the utilization value is 1/3 (less than (2/ (0.4+1))-1 in VM_1), it can be assigned on VM_1 too. For task 321, the utilization value is 0.2 (greater than 2/ (0.4+1)*(0.333+1)-1 in VM_1), it can be assigned on a new VM (VM_2). When the algorithm returns, all the five tasks can be scheduled on two VMs, Fig.3 shows the result.

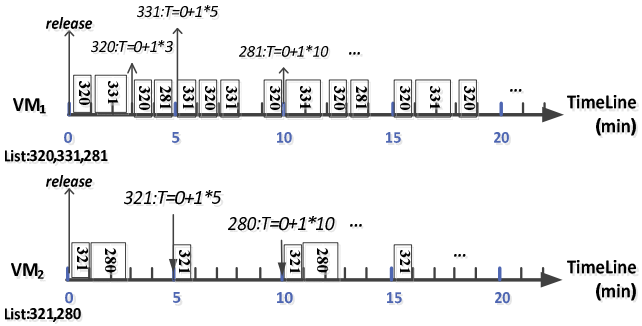

Fig.3. A scheduling instance

V. CASE-STUDY AND EVALUATION

In this section, we will illustrate the effectiveness of the scheduling *F*ramework for *P*eriodic *T*asks (*FPT*) by a case-study and its evaluation. We start with simulations of a large geo-distributed datacenters based on workload traces from iVCE test bed. We then use a smaller set of micro simulations to evaluate the efficiency.

As our iVCE test bed has been used in real production, there is a certain risk to conduct repeatable large-scale experiments on the real infrastructure. Therefore, the CloudSim toolkit [21] has been chosen as a simulation platform as it is a modern simulation framework aimed at cloud computing environments.

We have simulated 12 data centers according to the workload trace. Each datacenter is composed of 480 VMs those can be divided into 6 types (3 network operators and 2 ways of network access). Each VM is modeled to have one CPU core with the performance equivalent to 2000 MIPS, 2 GB of RAM and 8 GB of storage.

A. *Efficiency of alleviating the overhead*

Fig.4 shows one-day trace of arrived tasks submitted by an identical user. The total number of tasks instances is about 100 thousands. For every instance, the scheduler needs to find an optimal machine to execute the task, leading to high overhead. Most of these tasks are actually periodic. Our simulation result in table 3 shows that the number of scheduling times can be reduced by more than 90% in *FPT*.

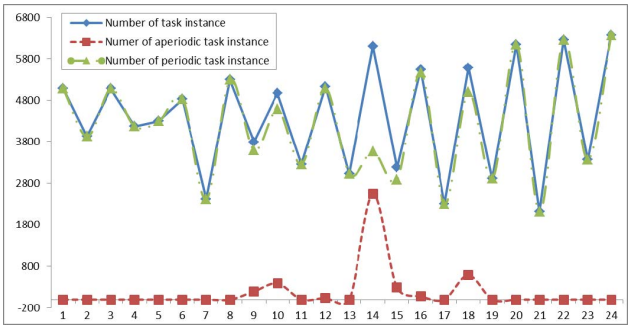

Fig.4. One-day trace of arrived tasks

TABLE 3. SCHEDULER OVERHEAD

	Testbed	***FPT***
Number of Scheduling	**105132**	**8125**

B. *Performance Evaluation*

Now we evaluate the performance of *FPT* using trace-driven simulations. In above workload traces, there are 101010 periodic task instances those need be scheduled to execute on 12 geo-distributed data centers. For each data center, we make a micro simulation to evaluate the scheduling success ratio. If the scheduler has allocated the resources and generated task instances execution sequence rightly for a task, we call the task has been scheduled successfully.

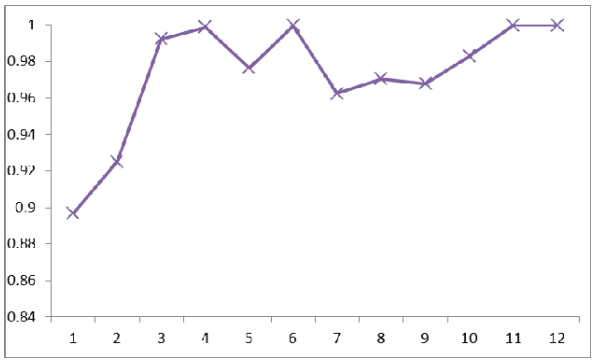

Fig.5. Scheduling success ratio of 12 datacenters

Fig.5 shows our simulation results. The average scheduling success ratio is 94.1%. The main cause of scheduling failure is task preemption. Highly frequent occurrences of preemption will consume system resources under very high loads, which is neglected in *FPT*. However, the ratio is much higher than our production test bed.

Though, the production environment is more complicated than the simulations, the current result of scheduling overhead and success ratio reflects a preliminary performance of our algorithm, but it has already shown the effectiveness of our framework. More comprehensive evaluation based on an actual test bed will be conducted in the future.

VI. RELATED WORK

This work is related to research in the following fields.

Task Scheduling in Data Centers. A large body of research has examined variants of task scheduling algorithm or framework in data centers (or cloud computing). Most of these works focus on lifting the efficiency. Ran S et al. present a provably-efficient online algorithm, GreFar, for scheduling batch jobs among multiple geographically distributed data centers [2]. GreFar minimizes the energy-fairness cost while providing queueing delay guarantees. Xu H et al. propose to make workload management for geo-distributed datacenters temperature aware and formulated the problem as a joint optimization of request routing and capacity allocation [5]. Zhang Z et al. present Fuxi, a distributed resource management and job scheduling system at Alibaba. There are three novel techniques that allow Fuxi to tackle the scalability and fault tolerance issues at Internet scale [12]. To the best of our knowledge, none of these woks have used the temporal characteristic of tasks to lift the efficiency.

Real-time tasks scheduling. Real-time tasks scheduling is a key problem in real-time system, which has decades of history. A scheduling algorithm which assigns priorities to tasks in a monotonic relation to their request rates (RM) is shown to be optimum among the class of all fixed priority scheduling algorithms in [20]. Oh Y et al. seek to minimize the total number of processors required to execute a set of periodic tasks such that deadlines are guaranteed by the Rate-Monotonic (RM) algorithm on each processor, an algorithm called RM-FFDU is given for this problem [15]. Bertogna M et al. addresses the schedulability problem of periodic and sporadic real-time task sets with constrained deadlines on a multiprocessor platform and a sufficient schedulability algorithm that are able to check whether a periodic or sporadic task set can be scheduled is presented in [17]. All of these works offer the theoretical foundation for our research work in this paper.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have presented a scheduling Framework for Periodic Tasks (*FPT*), which aims to reduce the unnecessary scheduling for periodic tasks in Geo-Distributed data centers. The number of tasks those need to be scheduled is the most essential factor for determining the pressure of the scheduler. Reducing the number of

scheduling times is most simple and intuitionistic. In some applications, tasks are dispatched periodically. It is feasible to realize one-time scheduling for multiple executions, alleviating the overhead of data centers. *FPT* is mainly composed of four modules. For a task set, an algorithm aiming to find the minimum number of VMs required to guarantee the scheduability and generate the task instance execution sequences on each VM is also presented. A case-study and its evaluation are given to show the efficiency of our framework.

Future work will concentrate on improving the scheduling algorithm for generating execution sequence of periodic tasks. The current version is based on Rate-Monotonic algorithm that belongs to preemptive scheduling. However, tasks preemption may decrease the scheduling success ratio. We also plan to work on other features such as bringing moderate fault tolerance into *FPT*, fine tuning algorithms to realize hybrid scheduling of periodic and aperiodic tasks, and alleviating the imbalance of resource utilization.

ACKNOWLEDGMENT

This work was supported by a grant from the Major State Basic Research Development Program of China (973 Program) (No. 2011CB302605).

REFERENCES

[1] https://www.google.com/about/datacenters/inside/locations.

[2] Ren S, He Y, Xu F. Provably-efficient job scheduling for energy and fairness in geographically distributed data centers[C]//Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on. IEEE, 2012: 22-31.

[3] Chen C, He B, Tang X. Green-aware workload scheduling in geographically distributed data centers[C]//CloudCom. 2012: 82-89.

[4] Liu Z, Lin M, Wierman A, et al. Greening geographical load balancing[C]//Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems. ACM, 2011: 233-244.

[5] Xu H, Feng C, Li B. Temperature aware workload management in geo-distributed datacenters[C]//Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems. ACM, 2013: 373-374.

[6] Goudarzi H, Pedram M. Geographical Load Balancing for Online Service Applications in Distributed Datacenters[C]//Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on. IEEE, 2013: 351-358.

[7] Wang W, Li B, Liang B. Dominant resource fairness in cloud computing systems with heterogeneous servers[J]. arXiv preprint arXiv:1308.0083, 2013.

[8] Sharma B, Chudnovsky V, Hellerstein J L, et al. Modeling and synthesizing task placement constraints in Google compute clusters[C]//Proceedings of the 2nd ACM Symposium on Cloud Computing. ACM, 2011: 3.

[9] Bialecki A, Cafarella M, Cutting D, et al. Hadoop: a framework for running applications on large clusters built of commodity hardware[J]. Wiki at http://lucene. apache. org/hadoop, 2005, 11.

[10] Hindman B, Konwinski A, Zaharia M, et al. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center[C]//NSDI. 2011, 11: 22-22.

[11] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[12] Zhang Z, Li C, Tao Y, et al. Fuxi: a Fault-Tolerant Resource Management and Job Scheduling System at Internet Scale[J]. Proceedings of the VLDB Endowment, 2014, 7(13).

[13] Apache Software Foundation. Apache Spark Lightning-Fast Cluster Computing.http://spark.apache.org/

[14] http://en.wikipedia.org/wiki/Real-time_operating_system

[15] Oh Y, Son S H. Fixed-priority scheduling of periodic tasks on multiprocessor systems [J]. Department of Computer Science, University of Virginia, Tech. Rep. CS-95-16, 1995: 1-37.

[16] Andersson B, Jonsson J. Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition[C]//Real-Time Computing Systems and Applications, 2000. Proceedings. Seventh International Conference on. IEEE, 2000: 337-346.

[17] Bertogna M, Cirinei M, Lipari G. Schedulability analysis of global scheduling algorithms on multiprocessor platforms [J]. Parallel and Distributed Systems, IEEE Transactions on, 2009, 20(4): 553-566.

[18] Devi U M C, Anderson J H. Tardiness bounds under global EDF scheduling on a multiprocessor [J]. Real-Time Systems, 2008, 38(2): 133-189.

[19] Srinivasan A, Baruah S. Deadline-based scheduling of periodic task systems on multiprocessors [J]. Information Processing Letters, 2002, 84(2): 93-98.

[20] Liu C L, Layland J W. Scheduling algorithms for multiprogramming in a hard-real-time environment[J]. Journal of the ACM (JACM), 1973, 20(1): 46-61.

[21] Calheiros R N, Ranjan R, Beloglazov A, et al. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms[J]. Software: Practice and Experience, 2011, 41(1): 23-50.