Multicloud-Based Evacuation Services for Emergency Management

MCES: Multi-cloud Based Evacuation Services for Emergency Management

Mianxiong Dong, He Li, Kaoru Ota, Laurence T. Yang, and Haojin Zhu

***Abstract*—A smart evacuation needs a scalable and flexible system to provide service in both emergency and normal situation. A single cloud service is usually limited to support scaling up requirement in emergency especially with a large geographic scope. In this article, we propose MCES, a multi-cloud architecture that deploy the smart evacuation services in multiple cloud providers, which can tolerant larger pressure than single cloud-based services. Usually, this system maintains basic service to support monitoring, while in emergency, the visits to the service will scale up enormously. This means MDSE supports fast scaling up service capacity in a short time. We use a three layer cloud instance management to support rapid capacity scaling up in MCES. By conducting extensive simulations, we demonstrate that our proposed MCES significantly outperforms single cloud solutions under various emergency settings.**

***Index Terms*—Evacuation Services; Emergency Management; Multiple Cloud.**

I. Related Works

A. Cloud-based Evacuation Services

There some existing works propose prototypes that adopt cloud computing in the evacuation system. Chu et al. [1] present a hybrid building fire evacuation system with mobile phones and cloud computing. In their prototype, they put the rout computing into the cloud service and use mobile phone to get the sensor information. However, they only move the evacuation services from local to the cloud simply without any consideration about the emergency management.

Chen et al. [2] present a system about rescue service with mobile cloud computing. From their work, the rescue system is similar with evacuation system which needs a rescue service to find the evacuation route and also need the information from sensors. As the prototype in their work, they also focus on the mobile client rather than the cloud services.

Ahn et al. [3] propose another mobile evacuation system named RescureMe. They provide a video-based evacuation interface and also adopt the cloud service to calculate the exit route. Similar with other works, even though they design their work on cloud services, they consider the cloud resources as same as the local servers.

Ling et al. [4] propose a cloud service-oriented visual prototype system for regional crowd evacuation. They add the service-oriented idea to their prototype and consider the scenario about evacuation as cloud services. Even though their modeling include the cloud-oriented factor, it is hard to find the different with the traditional evacuation systems.

B. Multi-cloud Service

Multiple cloud service is a service based more than one cloud platform. Since the limitations with single cloud platform, many existing work focus on the scheduling between multiple cloud platform. Chaisiri et al. [5] propose an optimal algorithm to placement virtual machine to multiple cloud providers to minimize the cost spending. In their modeling, enough though they analyze different payment planes with various demands, they only consider the constraints statically, which is not an appropriate method for the evacuation scenario.

Many similar research present many modeling with different constraints and requirement on the scheduling with multiple cloud platforms. Bossche et al. [6] study a modeling to minimize cost of external provision in a hybrid clouds. They focus on different workloads including deadline-constrained and non-migratable, and incorporate different resources in a binary integer programming problem formulation. Breitgand et al. [7] also propose some integer programing formulations for placement of

Mianxiong Dong and Kaoru Ota are with Muroran Insitute of Technology, Japan.

He Li is with Huazhong University of Science and Technology, China.

Laurence T. Yang is with St Francis Xavier University, Canada.

Haojin Zhu is with Shanghai Jiao Tong University, China.

VM workloads with a cloud as well as across multiple clouds collaborating. Meanwhile, they also provide a framework prototype combined with policies for load balancing and consolidation. In their prototype, they also develop a 2-approximation for scalability based on linear rounding. Ferreto et al. [8] study different greedy algorithm to placement VMs between various server consolidation schemes and give an experimental evaluation to demonstratethat greedy algorithms perform well in terms of number of physical machines used and VMs migrated. These work focus on the general scenario of VM placement without consideration about the price alters sharply between different statuses of instances. Meanwhile, without the consideration about the workload sudden increase, it is hard to apply their models to the evacuation service in emergency.

II. Introduction

With the rapid development of the cloud comptuing, many services are migrated to the cloud for better scalability, cost efficiency and other benefits [9] [10] [11]. Cloud computing is also an appropriate environment for deployment of evacuation services. Some cloud-based evacuation services move the main computing and numerous data to the cloud and provide services to users [12] [1].

If the evacuation system utilizes cloud computing as an external system performing calculation to determine the location of the client and then provide a possible evacuating route, the client can receive the estimated location and evacuating route to display with very short time. In addition, the system can provide calculation for all the people in the disaster place to determine their locations simultaneously, and assign various evacuating route to avoid congestion. The information of the disaster scene can also be shared with disaster relief.

Even though the cloud-based evacuation system can provide better service than before, maintaining full service capacity in the cloud platform to response emergent situations especially the large-scale disaster is expansive. As a result, the cloud-based evacuation system usually use a part of service instances to monitor the sensors and provide some basic services. When the disaster happens, the evacuation system will launch as many service instances as possible to meet the potential pressure to providing evacuation services for a large number of users. Unfortunately, since the I/O limitation in most cloud providers, it is hard to launch enough instances concurrently in a short period with single cloud platform [13]. It means the cloud-based evacuation system needs to maintain more active instances in normal times.

To focus that issue, we propose MCES, a multiple cloud platforms based evacuation system, which can provide better management than single cloud with lower cost in emergency. Different from single cloud architecture, MCSE uses computing resources from multiple cloud platform [14] or providers, which can provide better scalability and concurrency. Therefore, in emergency, MCSE can launch much more cloud instances concurrently between multiple cloud platform than single cloud evacuation system. As a result, the normal maintenance only needs to rent the active instance for the basic services.

Based on the proposed design, we then study how to deploy instances to a set of cloud platforms such that capacity service is maximum in emergency. This deployment problem has several challenges. First, each cloud may have a different capacity of instances. We need to deploy instances no more than the capacity of each cloud platform. Second, the concurrency to launch instances is also different between cloud platforms. To guarantee the response quickly in emergency, we need to consider the concurrency of each cloud platform when deploy the standby instances. Third, the number of active instances need to meet the requirement of the maintenance and the cost of active instances are much expensive than standby instances. We need to deploy instances with limited budget with enough active instances.

The main contributions of this paper are summarized as follows.

- First, we propose a multiple cloud evacuation architecture, in which the service instances are deployed in multiple cloud platforms.
- Second, we study a instance deployment problem to maximum the service capacity in emergency while satisfying the service budget, in terms of the required maintenance capacity of active instances, the capacity and concurrency of each platform. We propose an efficient algorithm to solve the controller assignment with a good result ratio with the optimal assignment.
- Finally, we evaluate our work with experiments on a prototype and simulations.

The rest of this paper is summarized as follows. Section I reviews the related work. Section III presents the system design and problem formulation. An efficient algorithm is proposed in Section IV. Section V gives the simulation results. Finally, Section VI concludes this paper.

III. Framework and Problem Statement

In this section, we fist brief MCES framework and the instance statuses in the cloud platform. Then, we state the instance deployment in MCES for emergency management.

A. MCES Framework

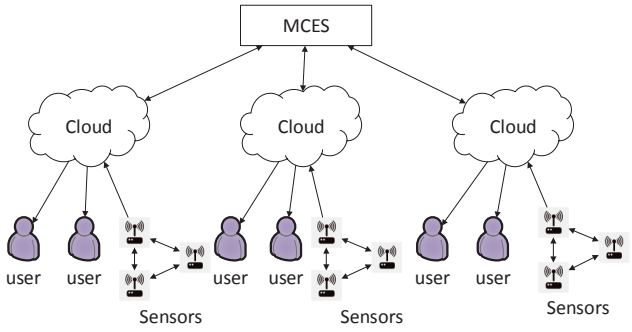


Fig. 1: Motivation of MCES

As shown in Fig. 1, we briefly introduce the MCES framework. For better flexibility for the evacuation services, we adopt the cloud model names *Infrastructure as a service* (IaaS) in which we can deploy evacuation services to the powerful instances provided by the cloud platform. Therefore, the MCES framework deploys and managements instances in multiple cloud platforms. The instances provide services to users and monitor the sensors to analyze the situation of the environment both in normal times and emergency. The users access these instances directly after a simple authentication in the service portal.

When the disaster happens, the MCES will launch the standby instances in all cloud platforms to meet potential pressure form a large number of users. These service instances response the access and analyze the evacuate route for each users as soon as possible with the data from sensors. While in normal times, it is no need to maintain too much active instances for service maintenances and sensor monitoring. Therefore, MCES will sleep or delete most instances launched in emergency and only maintain a small number of active instances for the basic services.

Even though the evacuation service store little personal information and other private data, we add a security module in each instance to protect some potential leakage or other attack. The security module includes a user authentication and user space isolation. Before the service begins, all users need to take a simple step to log in to the center management of MCSE. All user information is management by centralized management server. To isolate the user space, we use a sandbox mechanism in each instance. For each user, accessible services are encapsulated in single sandbox.

B. Instance Statuses

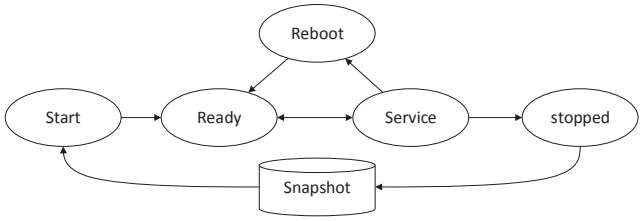


Fig. 2: Status transitions of the cloud instances

Since the MCES will transit instances between normal times and emergency, we describe the instance statuses in cloud platforms. Usually, in existing cloud platform, each instance have several statuses including Start, Ready, Service, Reboot, Stopped, Snapshot, etc., as shown in Fig. 2. When an instance created, it is in the Start status, which means it is start the basic functions like the operating system or other modules. This status will take a short period from tens of seconds. After start status, the instance enters the status of Ready to wait for the services beginning. After the needed services start, the instance enters the Service status for providing services. When some problems happen and the instance need to reset its services, the status will be transited to Reboot and after tens of second become to Ready. If the service need to be stopped, the instance will be shut down then transited to stopped status. In many cloud platforms, if a instance stopped, it will delete the data of this instance to release the resource. To focus this procedure,

tenants will backup the data of the stopped instance as a snapshot stored in the cloud platform.

In these statuses, there are only three stable statuses, Ready, Service and Snapshot. These statuses of instances means different cost. Instances in the snapshot status have the lowest fee since it only needs storage space to store the snapshot. Instances in Ready status have less cost than those in Service status. Usually, the different of cost between Ready and Service is much smaller than the different between Ready and Snapshot. For example, in google compute platform, the snapshot storage cost 0.125 dollars per GB/Month while a standard instance needs from 32 to 508 dollars. Meanwhile, the instances usually needs more budget on the cost of CPU time and disk I/O. Therefore, to maintain the lowest cost after Start status, MCES will turn the instances in Ready status to sleeping. In the rest of this paper, we use sleeping instead of Ready status to denote the instances wait for services beginning.

C. The IDME Problem

TABLE I: Notations in the state cycle problem

Notation	Description
C	Clouds in the network
c_i	Cloud i
$\mathbb{B}_i$	Capacity of cloud i
T_i	Latency between cloud i and user
D_{ri}	Instance running cost in cloud i
D_{si}	Instance sleep cost in cloud i
D_{di}	Instance snapshot cost in cloud i
L_{sr}	Instance waking time in cloud i
L_{ds}	Instance creation time in cloud i
K_{si}	Instance waking concurrency of cloud i
K_{di}	Instance creation concurrency of cloud i
S_i	Instance service capacity in cloud i
X_i	Service node number in cloud i
Y_i	Sleep node number in cloud i
Z_i	Snapshot node number in cloud i
$\mathbb{D}$	Budget of the evacuation services
$\mathcal{L}$	Total latency
$\mathcal{D}$	Service maintenance cost
$\mathbb{L}$	Required maximum response time in emergency
$\mathbb{B}_i$	Instance number of cloud i
$\mathbb{S}_m$	Service capacity for normal maintenance

We consider a MCES framework as show in Fig. 3

We use three values, X_i, Y_i and Z_i, to denote the three different number of instances in three status.

Therefore, we define $\mathbb{D}$ to denote the budget of the evacuation service shown in (3).

Since the evacuation service need to response the emergency quickly, as shown as (1), we defined $\mathbb{L}_r$ to denote the maximum time to transit sleeping instances and snapshots to running instances for the requirement in emergency.

$$\max_{i=1,2,...,|C|}[(\lceil\frac{Y_i}{K_{si}}\rceil+\lceil\frac{Z_i}{K_{si}}\rceil)L_{sr}+\lceil\frac{Z_i}{K_{di}}\rceil L_{ds}]\leq\mathbb{L} \quad (1)$$

We use $N_{si}\leftarrow\frac{L_{sr}}{K_{si}}$ and $N_{di}\leftarrow N_{si}+\frac{L_{ds}}{L_{di}}$ to simplify (1) as (4).

Meanwhile, we can not deploy instances exceed the capacity of the cloud service. We use $\mathbb{B}_i$ to denote the maximum service capacity of cloud i as shown in (5).

In the normal time, for maintains of the data collection, sensor management and other processing, it needs some running instances in the evacuation service. To simplify the problem, we use simple model [15] to describe the environment for sensor located. Therefore, , we use $\mathbb{S}_m$ to denote request for maintain the normal service and the minimum service capacity should satisfy (6).

With multiple cloud platform, we can get a larger service capacity than single cloud environment. We use $\mathcal{S}_t$ to denote the total service capacity of MCES shown in (2).

$$\text{Maximize:} \quad \mathcal{S}_t=\sum_{i=1}^{|C|}[(X_i+Y_i+Z_i)S_i] \quad (2)$$

$$\text{Subject to:} \quad \sum_{i=1}^{|C|}(X_iD_{ri}+Y_iD_{si}+Z_iD_{di})\leq\mathbb{D} \quad (3)$$

$$\sum_{i=1}^{|C|}(Y_iN_{si}+Z_iN_{di})\leq\mathbb{L} \quad (4)$$

$$X_i+Y_i+Z_i\leq\mathbb{B}_i \quad (5)$$

$$\sum_{i=1}^{|C|}(X_iS_i)\geq\mathbb{S}_m \quad (6)$$

The problem of instance deployment in MCES for emergency management (IDME): given a set of cloud platforms, the IDME problem attempts to built maximum service capacity in emergency by deploy instances from multiple cloud platforms with limited budget. Meanwhile, this service can support normal maintains and response emergency quickly with enough average QoS.

D. Hardness analysis

Theorem 1: The instance deployment problem is NP-hard.

Proof: **The bounded knapsack problem problem**: given a set of item kinds $\{a_1, a_2, ...a_n\}$, each kind of items a_i with a value v_i and a weight w_i, and the maximum weight that we can carry in the bag is W ($W < \sum_{i=1}^{n} w_i$), is there a knapsack scheme such that maximum the sum of values of the items in the bag and the sum of weight must be no more than W while the number of each kind of items is no more than c_i? ■

For each item a_i with a value v_i and weight w_i, we create a cloud platform with a cost set $D_{ri} = w_i$, $D_{si} = 0$ and $D_{di} = 0$, a value $S_i = v_i$, a instance waking time $L_{sr} > \mathbb{L}$, a instance creation time $L_{ds} > \mathbb{L}$, a capacity $B_i = c_i$, and a maintain capacity require $\mathbb{S}_m = 0$. Therefore, the constraints and the service capacity of the IDME problem are as flows.

$$\text{Maximize:} \quad \mathcal{S}_t = \sum_{i=1}^{n} X_i v_i \tag{7}$$

$$\text{Subject to:} \quad \sum_{i=1}^{n} X_i w_i \leq \mathbb{D} \tag{8}$$

$$X_i \leq c_i \tag{9}$$

We first suppose a solution that the bounded knapsack problem that we choose a set $\{X_1, X_2, ...X_n\}$ of items from $\{a_1, a_2, ...a_n\}$ and the sum of the item value is maximum. In the corresponding solution of the controller assignment problem, we choose X_i instance from each cloud platform C_i, and the total cost of the instances is less than W.

We then suppose that the instance deployment problem has a solution that a set of X_i instances is selected from each cloud platform C_i. From the equation (8) and (7), the set of X_i forms a solution of the bounded knapsack problem.

It is easy to see that the instance deployment problem is in NP class as the objective function associated with a given solution can be evaluated in a polynomial time. Thus, we conclude that the controller assignment problem is NP-hard.

IV. Solving the IDME problem

In this section, we propose an algorithm, called instance status cycle deployment (ISCD) algorithm, to solve the IDME problem. Its basic idea is to organize the cloud instances in three layers includes service, sleep and snapshot shown in figure 3. With these three layers, we can use different strategies to make a leverage to balance the time complex and algorithm performance.

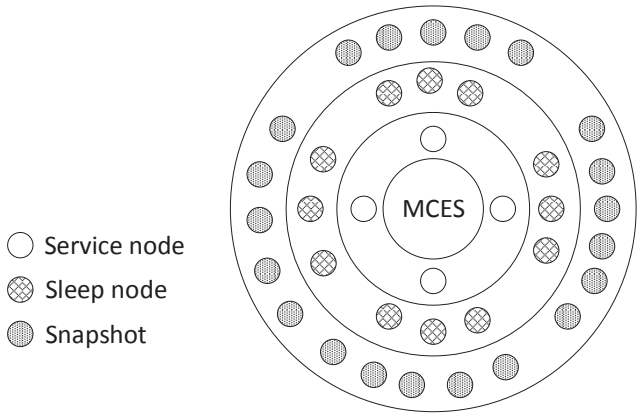


Fig. 3: Status cycle scheduling

For the service layer, considering the cost of the service instances is most expensive than those in other two status, the algorithm deploys the instances in the service level first. Considering the $X_i \ll Y_i$ and $X_i \ll Z_i$ usually, we use a greedy strategy to choose the most cost efficiency instance for the maintain service.

Algorithm 1 Greedy deployment for the service layer

```
Sort cloud platform in C' = {c_Π1, c_Π2, .., c_Π|C'|}
    such that D_rΠ1/S_Π1 ≤ D_rΠ2/S_Π2 ≤ ... ≤ D_rΠ|C'|/S_Π|C'|;
for i from 1 to |C'| do
    X_i ← 0, Y_i ← 0 and Z_i ← 0;
end for
i ← 1;
while Σ_{i=0}^{|C'|} (I_s S_i) < S_m do
    X_i ← 0;
    if X_i > B_i then
        i ← i + 1;
    end if
end while
```

In the sleep layer, since the required node are more than that in the service layer, the deployment will influent the performance more obviously. Meanwhile, low efficient deployment result in the sleep layer will also increase the response time. As mentioned in Section III, the IDME problem has four constraints in (3)(4)(5)(6). Fortunately, since the sleep nodes can not participate into the service maintenance, the constraint (6) will not affect the deployment in the sleep layer. But three constraints complicate the problem for higher efficient problem. Therefore, for leveraging the time complex and

algorithm efficiency, we choose another strategy based on simulated annealing. From the existing analysis of 0-1 knapsack problem [16], we choose a strategy based on the simulated anneling approach which is a suitable solution for knapsack problem with acceptable time complex. We define the energy function as follows.

$$E(Y) = \sum_{i=1}^{|C|} Y_i S_i$$

We also use a function $F(Y)$ to get the total latency of the deployment of Y.

Algorithm 2 Simulated annealing for the sleep layer

$Y_{sleep} \leftarrow \emptyset$;
for c_i in C **do**
  Y_i = $randInt(0, B_i - X_i)$;
  $Y_{sleep} \leftarrow Y_{sleep} \bigcup Y_i$;
end for
$E \leftarrow E(Y_{sleep})$;
$L \leftarrow L(Y_{sleep})$;
for k in $range(0, k_{max})$ **do**
  $Y'_{sleep} \leftarrow neighbour(Y_{sleep})$;
  $E' \leftarrow E(Y'_{sleep})$;
  $L' \leftarrow L(Y'_{sleep})$;
  if $E' > E$ and $L' < L$ **then**
    $Y_{sleep} \leftarrow Y'_{sleep}$;
  else if $E' > E$ and $L' \geq L$ and $L' < \mathbb{L}$ **then**
    $P \leftarrow e^{\frac{L-L'}{L}}$;
    if $randFloat(0, 1) > P$ **then**
      $Y_{sleep} \leftarrow Y'_{sleep}$;
    end if
  else if $E' < E$ and $L' < L$ **then**
    $P \leftarrow e^{\frac{E'-E}{E'}}$;
    if $randFloat(0, 1) > P$ **then**
      $Y_{sleep} \leftarrow Y'_{sleep}$;
    end if
  end if
end for

For the snapshot level, we design a dynamic programming to select instance from the cloud platforms. We use a function Snapshotpack to process each cloud i in all cloud platform. In function $Snapshotpack$, we use two sub functions, Subpack and Subsubpack to process the different value of constraints of cloud i. The Subpack function find Z_i for cloud i while Subsubpack find the value in $\log(B_i - X_i - Y_i)$ times. When the cost or response latency exceeds the constraint of the budget or the maximum response latency with the capacity of cloud i, the algorithm use Subpack function. Otherwise, the algorithm invoke Subsubpack function. Generally, since there are three loops in this algorithm, the time complex is $O(|C|\mathbb{D}\mathbb{L}\log B)$. Although it seems a little complex for running this algorithm, it is acceptable with its accuracy and limited instance number. To optimize this algorithm, we also set the unit of the latency to second and the the unit of budget to 100 dollars. Therefore, the time complex is $O(|C|\lceil\frac{\mathbb{D}}{100}\rceil\lceil\mathbb{L}\rceil\log B)$.

Algorithm 3 Dynamic programming for the snapshot layer

$\mathbb{D}' \leftarrow \mathbb{D} - \sum_{i=1}^{|C|}(X_i D_{ri} + Y_i D_{si})$;
$\mathbb{L}' \leftarrow \mathbb{L} - \sum_{i=1}^{|C|} Y_i N_{si}$;
for i from 1 to $|C|$ **do**
  SNAPSHOTPACK(D_{di}, S_i, N_{si} $B_i - X_i - Y_i$);
  $c, t \leftarrow \arg\max f$;
  $f[c][0] \leftarrow f[c][t]$;
end for
function SNAPSHOTPACK(d, s, n, b)
  if $db >= \mathbb{D}$ and $nb >= \mathbb{L}'$ **then**
    SUBPACK(d, s, n);
    return ;
  end if
  $k = 1$;
  while $k < b$ **do**
    SUBSUBPACK(kc, ks, kn);
    $b = b - k$;
    $k \leftarrow 2k$;
  end while
end function
function SUBPACK(d, s, n)
  for d' from d to $\mathbb{D}'$ **do**
    for n' from n to $\mathbb{L}'$ **do**
      $f[d'][n'] \leftarrow \max(f[d'][n'], f[d' - D_{di}][n' - N_{di}] + S_i)$;
    end for
  end for
end function
function SUBSUBPACK(d, s, n)
  for d' from c to $\mathbb{D}'$ **do**
    for n' from n to $\mathbb{L}'$ **do**
      $f[d'][n'] \leftarrow \max(f[d'][n'], f[d' - d][n' - n] + s)$;
    end for
  end for
end function

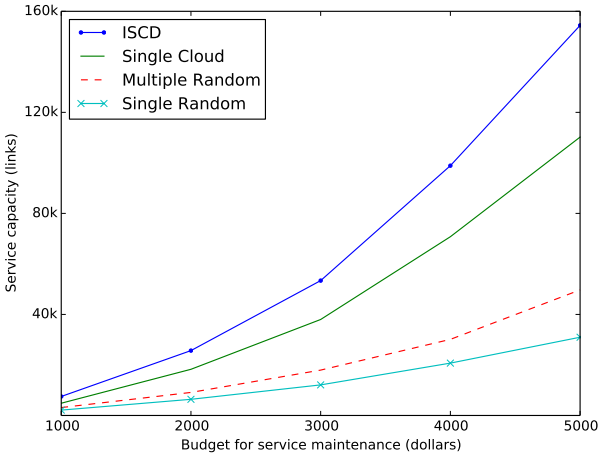


Fig. 4: Service capacity with different budget on service maintenance

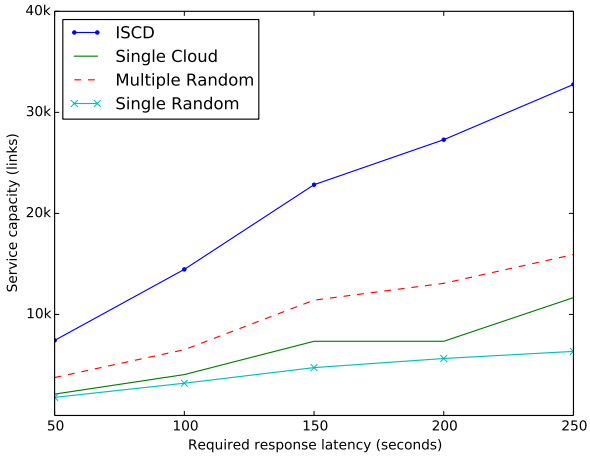


Fig. 5: Service capacity with different required response latency

V. Performance Evaluation

We conduct simulation-based experiments to evaluate the performance of the proposed algorithms. To evaluate the performance in general cases, we generate random networks and compare the cost of different controller assignment algorithms.

For the implementation and evaluation of the instance placement, we used the script language Python 2.7 and the numpry library, and test multiple workloads. We use the price data from 24 cloud platforms and test the status transition latency from the snapshot to active status and the latency from the active status to service status. Since it is hard to get the real-world concurrency data, we set the wakeup concurrency of each cloud platform is distributed evenly in range [1000, 2000] and the startup concurrency is distributed evenly in range [200, 700]. We also study some existing works based on cloud [1] then set the service capacity of each instance is distributed evenly in range [100, 200] links. Meanwhile, the response latency in emergency is set to 300 seconds which is enough for the people evacuation.

For comparison, the following two schemes are considered: (1) Single cloud with dynamic programming.

(2) Random deployment with a multiple cloud framework.

(3) Random deployment with a single cloud platform.

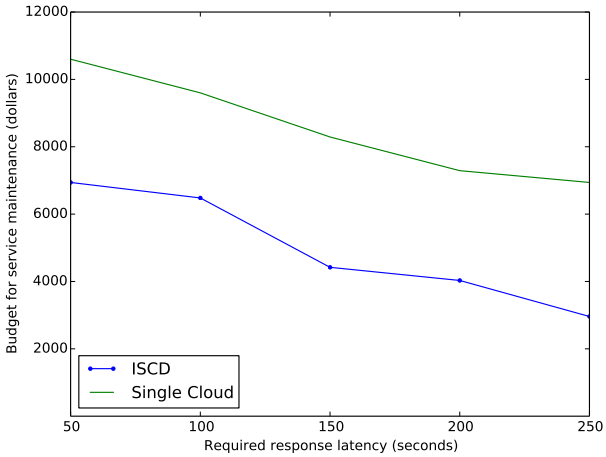


Fig. 6: Minimum budget for the evacuation service in the scenario of Padang

As shown in Fig. 4, we first study the maximum service capacity in emergency with different budget for service maintenance. The budget is set from 1000 dollars to 5000 dollars per month. With 1000 dollars, it seems limited budget is hard to maintain a large enough cluster for the evacuation service in emergency. With the increased budget, the service capacity in emergency is increased rapidly. The ISCD deployment performs best performance in all solutions. Since the latency requirement is no strict, the deployment on single cloud performs 80% capacity of the performance in MCES with ISCD deployment. The performance of two random deployment strategies is much lower than the dynamic

algorithm and ISCD deployment especially with big budget.

Then, we analyze the performance with strict response latency in emergency. We test the service capacity with the requirement of response latency from 50 seconds to 250 seconds and the budget of 2000 dollars. With limited response latency, it needs higher concurrency on instance startup. As a result, the ISCD deployment in MCES performs much better than the single cloud platform. With better concurrency, the random deployment in MCES even performs better then dynamic programming with single cloud platform. With limited concurrency, the dynamic programing performs just a little better than the random deployment in the single cloud platform.

With the initial evaluation of our solution, with the same budget for service maintenance, we find that our platform can provide better service capacity than single cloud platform in emergency. It because MCES can maintain less active instances in the normal times and use more snapshot and sleep instances with lower cost. When the disaster happens, with higher concurrency provided by multiple cloud platforms, MCES can startup much more instances than single cloud platform in the given response latency.

We also take a real world scenario based simulation to evaluate whether the MCES provide capacity for the evacuation in real disaster. We choose the data from the Indonesian city of Padang which faces high risk of being inundated by a tsunami ware. The city has more than 1,000,000 people in which 300,000 people living in the dangerous area. From some existing work, these people need more than 20 minutes for evacuation to the safe area and the warning time before the tsunami wave reaches the cost line is only 20-40 minutes [17]. It is more strict than the simulation setting with the limited tsunami warning time. We use the same setting as the input to the simulation. Considering the random deployment performs not enough in the past simulation, we only test the ISCD in MCES and the dynamic programming based placement in single cloud.

As shown in Fig.6, we analyze the budget with different response latency requirement for the evacuation service of the Padang scenario. From the result, we find if the MCES costs less than 7000 dollars per month, the evacuation service response time is less than 1 minutes which is much less than the evacuation time while single cloud based evacuation service costs near to 11000 dollars. If the time request becomes less strict, the budget needed by two systems are reduced rapidly. When the response latency request is more than 4 minutes, the budget taken by the MCES is less than 3000 dollars while the single cloud based service costs less than 7000 dollars. As a result, with the multi-cloud based deployment support, MCES costs much less than single cloud based service even in a real word scenario.

VI. Conclusion and Future Work

In this paper, we propose a multiple cloud based architecture that uses multiple cloud platforms to provide higher evacuation service capacity than single cloud platform in emergency. We design a framework named MCES to support the multiple cloud based evacuation services based on existing cloud platforms. In this framework, we can deploy instances in different statuses for lower cost. Based on this framework, we study a instance deployment problem to maximum the service capacity in emergency while satisfying the cost and response latency requirements of a given set of cloud platform. Finally, extensive simulations are conducted to show that the proposed algorithm can significantly maximum the service capacity in emergency.

In the future, we plan to implement a complete MCES framwork as a module in the opensouce cloud management OpenStack [18]. Meanwhile, it is signification to find appropriate deployment algorithm to suit the dynamic requirement. A deeper experiment with the real word testbed is also needed to evaluate the efficiency of the MCES framework.

Acknowledgment

This work is partially supported by JSPS KAKENHI Grant Number 25880002, 26730056, JSPS A3 Foresight Program, NSFC Grant No. 61450110085, No. 61272444, No. 61411146001, No. U1401253, No. U1405251, the Open Research Project of the State Key Laboratory of Industrial Control Technology, Zhejiang University, China (No. ICT1407).

REFERENCES

[1] L. Chu and S.-J. Wu, "An integrated building fire evacuation system with rfid and cloud computing," in *2011 Seventh International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP)*, Oct 2011, pp. 17–20.

[2] Y.-J. Chen, C.-Y. Lin, and L.-C. Wang, "Sensors-assisted rescue service architecture in mobile cloud computing," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2013, pp. 4457–4462.

[3] J. Ahn and R. Han, "Rescueme: An indoor mobile augmented-reality evacuation system by personalized pedometry," in *2011 IEEE Asia-Pacific Services Computing Conference (APSCC)*, Dec 2011, pp. 70–77.

[4] W. Ling, J. Wang, and X. Wei, "Cloud service-oriented modeling and simulation of regional crowd evacuation in emergency," in *Web-Age Information Management*, ser. Lecture Notes in Computer Science, Y. Chen, W.-T. Balke, J. Xu, W. Xu, P. Jin, X. Lin, T. Tang, and E. Hwang, Eds. Springer International Publishing, 2014, pp. 130–140.

[5] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, Dec 2009, pp. 103–110.

[6] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, July 2010, pp. 228–235.

[7] D. Breitgand, A. Marashini, and J. Tordsson, "Policy-driven service placement optimization in federated clouds," *IBM Research Division, Tech. Rep*, 2011.

[8] T. C. Ferreto, M. A. S. Netto, R. N. Calheiros, and C. A. F. De Rose, "Server consolidation with migration control for virtualized data centers," *Future Gener. Comput. Syst.*, vol. 27, no. 8, pp. 1027–1034, Oct. 2011.

[9] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[11] M. Dong, H. Lit, K. Ota, and H. Zhu, "Hvsto: Efficient privacy preserving hybrid storage in cloud data center," in *IEEE Conference onComputer Communications Workshops (INFOCOM 2014 WKSHPS)*, April 2014, pp. 529–534.

[12] Z. Alazawi, S. Altowaijri, R. Mehmood, and M. Abdljabar, "Intelligent disaster management system based on cloud-enabled vehicular networks," in *11th International Conference on ITS Telecommunications (ITST)*, Aug 2011, pp. 361–368.

[13] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding performance interference of i/o workload in virtualized cloud environments," in *IEEE 3rd International Conference on Cloud Computing (CLOUD 2010)*, July 2010, pp. 51–58.

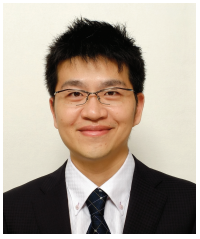
[14] I. Houidi, M. Mechtri, W. Louati, and D. Zeghlache, "Cloud service delivery across multiple cloud platforms," in *IEEE International Conference on Services Computing (SCC 2011)*, July 2011, pp. 741–742.

[15] M. Dou, J. Chen, D. Chen, X. Chen, Z. Deng, X. Zhang, K. Xu, and J. Wang, "Modeling and simulation for natural disaster contingency planning driven by high-resolution remote sensing images," *Future Generation Computer Systems*, vol. 37, no. 0, pp. 367 – 377, 2014.

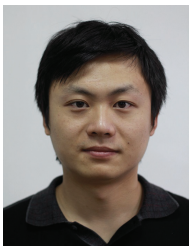
[16] A. Drexl, "A simulated annealing approach to the multiconstraint zero-one knapsack problem," *Computing*, vol. 40, no. 1, pp. 1–8, 1988.

[17] G. Lämmel, M. Rieser, K. Nagel, H. Taubenböck, G. Strunz, N. Goseberg, T. Schlurmann, H. Klüpfel, N. Setiadi, and J. Birkmann, "Emergency preparedness in the case of a tsunami–evacuation analysis and traffic optimization for the indonesian city of padang," in *Pedestrian and Evacuation Dynamics 2008*, W. W. F. Klingsch, C. Rogsch, A. Schadschneider, and M. Schreckenberg, Eds. Springer Berlin Heidelberg, 2010, pp. 171–182.

[18] K. Pepple, *Deploying OpenStack*. " O'Reilly Media, Inc.", 2011.



Mianxiong Dong received B.S., M.S. and Ph.D. in Computer Science and Engineering from The University of Aizu, Japan. He is currently an Assistant Professor with Department of Information and Electronic Engineering, Muroran Institute of Technology, Japan. Before join Muroran-IT, he was a Researcher with National Institute of Information and Communications Technology (NICT), Japan. He was a visiting scholar with BBCR group at University of Waterloo, Canada supported by JSPS Excellent Young Researcher Overseas Visit Program from April 2010 to August 2011. Dr. Dong is currently a research scientist with A3 Foresight Program (2011-2016) funded by Japan Society for the Promotion of Sciences (JSPS), NSFC of China, and NRF of Korea. His research interests include wireless sensor networks, vehicular ad-hoc networks and wireless security.



He Li received the BS and MS degrees from Huazhong University of Science and Technology in 2007 and 2009, respectively. Currently, he is a PhD student in School of Computer Science and Engineering, Huazhong University of Science and Technology, China. His research interests include cloud computing and software defined networking. He is a student member of the IEEE and the IEEE Communication Society.

Kaoru Ota received M.S. degree in Computer Science from Oklahoma State University, USA in 2008 and Ph.D. degree in Computer Science and Engineering from The University of Aizu, Japan in 2012. She is currently an Assistant Professor with Department of Information and Electronic Engineering, Muroran Institute of Technology, Japan. She serves as a Guest Editor of IEEE Wireless Communications, IEICE Transactions on Information and Systems, Editor of Peer-to-Peer Networking and Applications (Springer), International Journal of Embedded Systems (Inderscience) and Journal of Cyber-Physical Systems. Her research interests include wireless sensor networks, vehicular ad hoc networks, and ubiquitous computing.

Laurence T. Yang received the BE degree in Computer Science and Technology from Tsinghua University, China and the PhD degree in Computer Science from University of Victoria, Canada. He is a professor in the Department of Computer Science, St. Francis Xavier University, Canada. His research interests include parallel and distributed computing, embedded and ubiquitous/pervasive computing, big data. He has published more than 200 papers in various refereed journals (around 40% on IEEE/ACM Transactions and Journals, others mostly on Elsevier, Springer and Wiley Journals). His research has been supported by the National Sciences and Engineering Research Council, and the Canada Foundation for Innovation.

Haojin Zhu received his B.Sc. degree (2002) from Wuhan University (China), his M.Sc. (2005) degree from Shanghai Jiao Tong University (China), both in computer science and the Ph.D. in Electrical and Computer Engineering from the University of Waterloo (Canada), in 2009. He is currently an Associate Professor with Shanghai Key Laboratory of Scalable Computing and Systems, Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research interests include wireless network security and distributed system security.