Multicloud-Based Evacuation Services for Emergency Management

Mianxiong Dong, Muroran Institute of Technology, Japan
He Li, Huazhong University of Science and Technology, China
Kaoru Ota, Muroran Institute of Technology, Japan
Laurence T. Yang, St. Francis Xavier University, Canada
Haojin Zhu, Shanghai Jiao Tong University, China

The multicloud-based evacuation services (MCES) architecture maintains basic monitoring and maintenance services during times of normal activity but quickly scales up service capacity during an emergency.

Many services are migrating to the cloud for its better scalability, cost efficiency, and other benefits.[1–3] Cloud computing also offers an appropriate environment for deploying evacuation services. Some cloud-based evacuation services move the main computing tasks and most data to the cloud.[4,5]

An evacuation system can use the cloud to quickly perform calculations to determine the client's location and suggest an evacuation route, allowing it to respond to clients in emergency situations within a short time. In addition, a cloud-based system can calculate the locations of all people in a disaster area simultaneously and vary evacuation routes to avoid congestion. It can also share information about the disaster scene with disaster relief.

Although a cloud-based evacuation system can provide improved services, maintaining full-service capacity in a cloud platform to respond to emergency situations, especially large-scale disasters, is expensive. As a result, most cloud-based evacuation systems use service instances to monitor sensors and provide some basic services (see the sidebar for a discussion of related systems). When a disaster occurs, the evacuation system launches as many service instances as possible to meet the potential demand for evacuation services for a large number of users. Unfortunately, because most cloud providers have limited I/O capability,

 2325-6095/14/$31.00

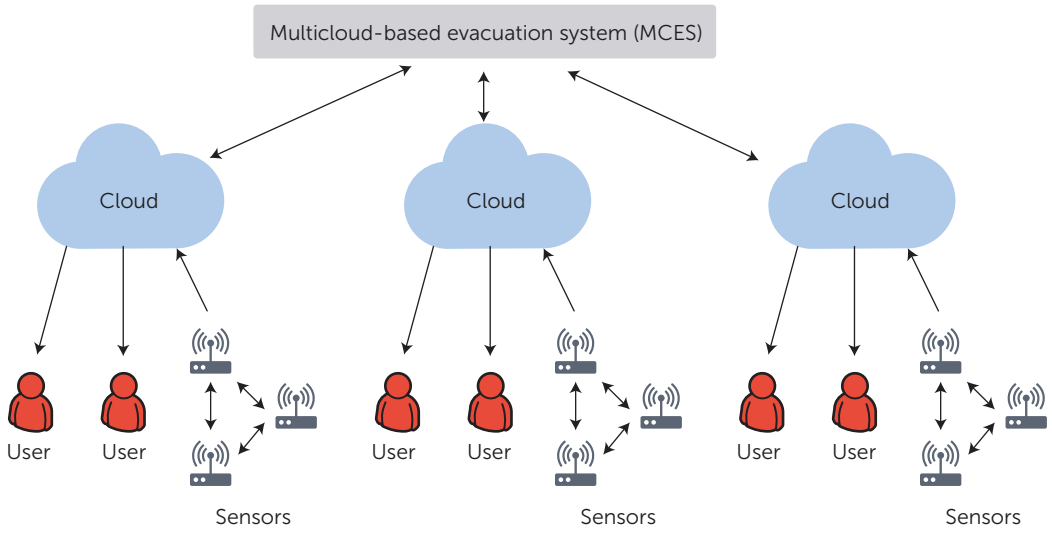

FIGURE 1. Overview of the multicloud evacuation system framework. MCES framework connects several cloud providers and these multiple providers interact with users and monitor the sensor networks.

launching enough instances concurrently in a short time period using a single cloud platform is difficult.[6] Thus, cloud-based evacuation systems must maintain more active instances in normal times.

Our multicloud-based evacuation system (MCES) provides better management than a single cloud system, with lower costs in an emergency. Unlike single cloud architectures, MCES uses computing resources from multiple cloud platforms[7] or providers, so that it can provide better scalability and concurrency. In an emergency, MCES can launch many more cloud instances concurrently than a single cloud evacuation system.

Using the proposed design, we study how to deploy instances to a set of cloud platforms to maximize service capacity in an emergency. This deployment problem has several challenges. First, each cloud might have a different capacity of instances, so we need to deploy no more instances than the capacity of each cloud platform. Second, the concurrency to launch instances differs among cloud platforms. To guarantee quick response in an emergency, we need to consider each cloud platform's concurrency when deploying standby instances. Third, we need to deploy enough active instances to meet maintenance requirements, keeping in mind the higher cost of active instances.

MCES Framework

Figure 1 gives an overview of the MCES framework. To provide more flexible evacuation services, we adopt an infrastructure-as-a-service (IaaS) cloud model, which lets us deploy evacuation services using instances provided by the cloud platform. Therefore, the MCES framework deploys and manages instances in multiple cloud platforms. These instances provide services to users and monitor sensors to analyze the situation in the environment in periods of normal times as well as emergencies. Users access these instances directly after a simple authentication in the service portal.

When a disaster occurs, the MCES launches the standby instances in all cloud platforms to meet the potential demand from a large number of users. Using sensor data, these service instances access users' locations and analyze their evacuation routes as quickly as possible. In periods of normal times, the system doesn't need to maintain many active instances for service maintenance and sensor monitoring. Therefore, MCES will put to sleep or delete most instances launched in an emergency, maintaining only a small number of active instances to provide basic services.

Although the evacuation service stores limited personal and other private data, we add a security

RELATED WORK IN CLOUD-BASED EVACUATION AND MULTICLOUD SERVICES

Several researchers propose prototype evacuation systems that adopt cloud computing. Liou Chu and Shih-Jung Wu present a hybrid building fire evacuation system with mobile phones and cloud computing.[1] In their prototype, they put routine computing tasks into the cloud service and use mobile phones to collect sensor information. However, they only move the evacuation services from local servers to the cloud without considering emergency management.

Yu-Jia Chen and his colleagues present a rescue service system that uses mobile cloud computing.[2] Their system is similar to an evacuation system, which finds the evacuation route based on sensor information and user location information. As with the prototype in their work, they focus on the mobile client rather than the cloud services.

Junho Ahn and Richard Han propose another mobile evacuation system, RescueMe.[3] They provide a video-based evacuation interface and adopt cloud services to calculate the exit route. As with other works, although they design their system for cloud services, they treat cloud resources the same as local servers.

Weiqing Ling and his colleagues propose a cloud service-oriented visual prototype system for regional crowd evacuation.[4] They add a service-oriented aspect to their prototype and consider a scenario in which evacuation is a cloud service. Although their model includes a cloud factor, it's hard to distinguish it from traditional evacuation systems.

Because of the limitations of single cloud platforms, many existing works focus on scheduling between multiple cloud platforms. Sivadon Chaisiri and his colleagues propose an optimal algorithm for placing virtual machines in multiple cloud providers to minimize cost.[5] In their model, although they analyze different payment plans with various demands, they only consider the constraints statically, which isn't an appropriate method for the evacuation scenario.

Many similar works present models with different constraints and requirements on scheduling with multiple cloud platforms. Ruben Bossche and his colleagues study a model to minimize cost of external provision in hybrid clouds.[6] They focus on different workloads including deadline-constrained and nonmigratable workloads, and incorporate different resources in a binary integer programming problem formulation. David Breitgand and his colleagues also

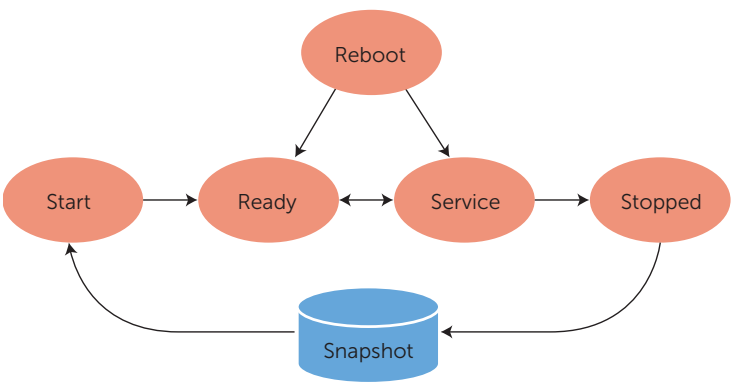

FIGURE 2. Status transitions of the cloud instances. Instances only provide evacuation services during the service status. After an instance stops and the user backs up it into a snapshot file, recovering it to service status takes a long time.

module in each instance to protect potential leakage or other attacks. The security module includes user authentication and user space isolation. Before the service begins, users must log in to the MCES management center. A centralized management server manages all user information. To isolate the user space, we encapsulate accessible services for each user in a single sandbox for each instance.

Instance Status

Because the MCES transits instances between normal and emergency periods, we describe the instance statuses in cloud platforms. In existing cloud platforms, instances typically have several statuses—start, ready, service, reboot, stopped, and snapshot—as Figure 2 shows. An instance begins in *start* status, which consists of basic functions and only lasts a short time (in the tens of seconds). Next,

propose some integer programing formulations for placing virtual machine (VM) workloads with a cloud as well as across multiple collaborating clouds.[7] Meanwhile, they also provide a framework prototype combined with policies for load balancing and consolidation. In their prototype, they also develop a two-approximation for scalability based on linear rounding. Tiago Ferreto and his colleagues study different greedy algorithms for placing VMs between various server consolidation schemes and give an experimental evaluation to demonstrate that greedy algorithms perform well in terms of number of physical machines used and VMs migrated.[8] These works focus on the general scenario of VM placement without consideration about how the price alters sharply between different statuses of instances. Meanwhile, without consideration about the workload sudden increase, it's hard to apply their models to the evacuation service in emergency.

References

1. L. Chu and S.-J. Wu, "An Integrated Building Fire Evacuation System with RFID and Cloud Computing," *Proc. 7th Int'l Conf. Intelligent Information Hiding and Multimedia Signal Processing* (IIH-MSP 11), 2011, pp. 17–20.
2. Y.-J. Chen, C.-Y. Lin, and L.-C. Wang, "Sensors-Assisted Rescue Service Architecture in Mobile Cloud Computing," *Proc. IEEE Wireless Comm. and Networking Conf.* (WCNC 13), 2013, pp. 4457–4462.
3. J. Ahn and R. Han, "RescueMe: An Indoor Mobile Augmented-Reality Evacuation System by Personalized Pedometry," *Proc. IEEE Asia-Pacific Services Computing Conf.* (APSCC 11), 2011, pp. 70–77.
4. W. Ling, J. Wang, and X. Wei, "Cloud Service-Oriented Modeling and Simulation of Regional Crowd Evacuation in Emergency," *Web-Age Information Management*, Y. Chen et al., eds., LNCS 8597, Springer, 2014, pp. 130–140.
5. S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal Virtual Machine Placement across Multiple Cloud Providers," *Proc. IEEE Asia-Pacific Services Computing Conf.* (APSCC 09), 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," *Proc. IEEE 3rd Int'l Conf. Cloud Computing* (Cloud 10), 2010, pp. 228–235.
7. D. Breitgand, A. Marashini, and J. Tordsson, *Policy-Driven Service Placement Optimization in Federated Clouds*, tech. report, IBM Research Division, 2011.
8. T.C. Ferreto et al., "Server Consolidation with Migration Control for Virtualized Data Centers," *Future Generation Computer Systems*, vol. 27, no. 8, 2011, pp. 1027–1034.

the instance enters *ready* status, where it waits for services to start. When the required services start, the instance enters the *service* status. If a problem occurs and the instance needs to reset its services, the status changes to *reboot*, and, after tens of second, it becomes *ready*. If the service needs to be stopped, the instance shuts down and changes to *stop* status. In many cloud platforms, if an instance stops, it deletes the instance data to release the resource. After this procedure, tenants usually back up the data of the stopped instance as a *snapshot* stored in the cloud platform.

Only three statuses—ready, service, and snapshot—are stable, and each status has a different cost. Instances in snapshot status have the lowest cost because they only need storage space. Instances in ready status have lower cost than those in service status. The cost difference between ready and service status is typically much smaller than the cost difference between ready and snapshot status. For example, in the Google Compute Platform, snapshot storage is $0.125 per Gbyte per month, so a standard instance needs from $32 to $508 per month. Meanwhile, instances usually need bigger budgets for CPU time and disk I/O. Therefore, to maintain the lowest cost for instances after they leave start status, MCES puts instances in ready status to the ready status. Considering an instance in read status like sleeping, in the remainder of this article, we'll use "sleeping" instead of ready status.

The Instance Deployment Problem

Figure 3 illustrates the status cycle scheduling for our multicloud-based system. We use three values, X_i, Y_i, and Z_i, to denote the number of instances in service, sleep, and snapshot status. We define $\mathbb{D}$ as

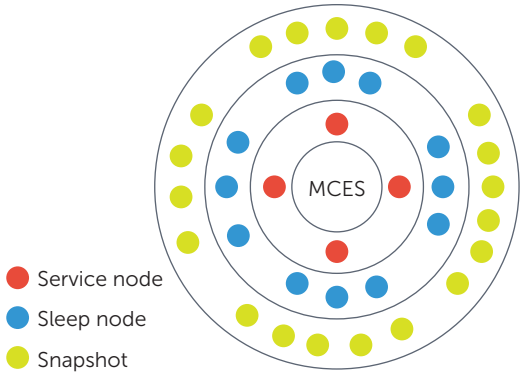

FIGURE 3. Status cycle scheduling in the MCES framework. MCES organizes all instance in three layers respectively with their different status: service, sleeping, and snapshot.

the budget of the evacuation service, as shown in Equation 3. Because the evacuation service needs to respond to emergencies quickly, we define $\mathbb{L}_r$ as the maximum time to transit sleeping instances and snapshots to running instances for the requirement in emergency, where

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

We use $N_{si}\leftarrow L_{sr}/K_{si}$ and $N_{di}\leftarrow N_{si}+L_{ds}/L_{di}$ to simplify Equation 1 as Equation 4 (see Table 1 for the full list of notations used in this article).

Meanwhile, we can't deploy instances in excess of the cloud service's capacity. We use $\mathbb{B}_i$ to denote the maximum service capacity of cloud i as shown in Equation 5.

For normal times, to maintain the data collection, sensor management, and other processes, MCES needs to maintain some running instances in the evacuation service. To simplify this problem, we use a simple model to describe the environment in which the sensor is located.[8] We use $\mathbb{S}_m$ to denote the request to maintain normal service, and the minimum service capacity should satisfy Equation 6.

With multiple cloud platforms, we can achieve a larger service capacity than in a single cloud environment. We use S_t to denote the total service capacity of MCES:

Maximize

$$S_t=\sum_{i=1}^{|C|}\left[\left(X_i+Y_i+Z_i\right)S_i\right] \tag{2}$$

Subject to

$$\sum_{i=1}^{|C|}\left(X_iD_{ri}+Y_iD_{si}+Z_iD_{di}\right)\leq\mathbb{D} \tag{3}$$

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

$$X_i+Y_i+Z_i\leq\mathbb{B}_i \tag{5}$$

$$\sum_{i=1}^{|C|}\left(X_iS_i\right)\geq\mathbb{S}_m\,. \tag{6}$$

Given a set of cloud platforms, the instance deployment in MCES for emergency management (IDME) problem attempts to build maximum service capacity in an emergency by deploying instances from multiple cloud platforms with a limited budget. Meanwhile, this service supports routine maintenance and responds to emergency quickly with sufficient average quality of service (QoS).

Hardness Analysis

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

The bounded knapsack problem is, given a set of item types, $\{a_1, a_2, \ldots, a_n\}$, each item type a_i has a value v_i and a weight w_i. The maximum weight that we can carry in the bag is

$$W\left(W<\sum\nolimits_{i=1}^{n}w_i\right).$$

Is there a knapsack scheme such that the sum of the values of the items in the bag and the sum of their weights is no more than W when the number of each item type 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 $S_i=v_i$; an instance waking time $L_{sr}>\mathbb{L}$; an instance creation time $L_{ds}>\mathbb{L}$; a capacity $B_i=c_i$; and a maintainance capacity requirement $\mathbb{S}_m=0$. Therefore, the constraints and the service capacity of the IDME problem are as follows:

Maximize

$$\mathbb{S}_t=\sum_{i=1}^{n}X_iv_i \tag{7}$$

Subject to

$$\sum_{i=1}^{n}X_iw_i\leq\mathbb{D} \tag{8}$$

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

We first consider a solution to the bounded knapsack problem in which we choose a set $\{X_1, X_2, \ldots, X_n\}$ of items from $\{a_1, a_2, \ldots, a_n\}$, where the sum of the items' values is the maximum amount

the bag can hold. In the corresponding solution to the controller assignment problem, we choose the X_i instance from each cloud platform C_i, and the total cost of the instances is less than W.

We then assume that the instance deployment problem has a solution in which we select a set of X_i instances from each cloud platform C_i. From Equations 7 and 8, the set of X_i forms a solution to the bounded knapsack problem.

Clearly, the instance deployment problem is in the NP-hard class because the objective function associated with a given solution can be evaluated in a polynomial time. Thus, the controller assignment problem is NP-hard.

Algorithms for Instance Deployment

We propose the instance status cycle deployment (ISCD) algorithm to solve the IDME problem (Figure 4). It organizes cloud instances into service, sleep, and snapshot layers (Figure 3). Within these three layers, we can use different strategies to balance time overhead and algorithm performance.

For the service layer, because service instances are more expensive than sleep or snapshot instances, the algorithm deploys service instances first. Given that $X_i \ll Y_i$ and $X_i \ll Z_i$ usually, we use a greedy strategy to choose the most cost-efficient instance for the maintenance service.

Because there are more nodes in the sleep layer than those in the service layer, inefficient deployment in this layer lead to more time overhead. As mentioned earlier, the IDME problem has the four constraints illustrated in Equations 3 through 6. Fortunately, because sleep nodes can't participate in service maintenance, the constraint in Equation 6 won't affect the deployment in the sleep layer and the deployment problem in sleep layer has only three constraints. Therefore, to leverage the time overhead and algorithm efficiency, we use a strategy based on simulated annealing. As in the existing analysis of the instance deployment problem,[9] we choose a strategy based on simulated annealing that solves the knapsack problem with an acceptable time overhead. Figure 5 shows our algorithm for deploying instances in the sleep layer.

We define the energy function as

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

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

For the snapshot layer, we use dynamic programming to select instances from the cloud platforms. We use the Snapshotpack function to process each cloud i in all of the cloud platforms. In Snapshotpack, we use two subfunctions, Subpack and Subsubpack, to process the different value of constraints of cloud i. The Subpack function

Table 1. 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

1: Sort cloud platform in $C' = \left\{c_{\Pi_1}, c_{\Pi_2}, .., c_{\Pi_{|C'|}}\right\}$ such that
$$\frac{D_{r\Pi_1}}{S_{\Pi_1}} \le \frac{D_{r\Pi_2}}{S_{\Pi_2}} \le ... \le \frac{D_{r\Pi_{|C'|}}}{S_{\Pi_{|C'|}}};$$
2: **for** i from 1 to $|C'|$ **do**
3: $X_i \leftarrow 0$, $Y_i \leftarrow 0$, and $Z_i \leftarrow 0$;
4: **end for**
5: $i \leftarrow 1$;
6: **while** $\sum_{i=0}^{|C'|}\left(I_s S_i\right) < \mathbb{S}_m$ **do**
7: $X_i \leftarrow 0$;
8: **if** $X_i > \mathbb{B}_i$ **then**
9: $i \leftarrow i + 1$;
10: **end if**
11: **end while**

FIGURE 4. The instance status cycle deployment (ISCD) algorithm is a greedy deployment strategy used in the service layer.

```
1:  Y_sleep ← ∅;
2:  for c_i in C do
3:      Y_i = randInt(0, B_i − X_i);
4:      Y_sleep ← Y_sleep ∪ Y_i;
5:  end for
6:  E ← E(Y_sleep);
7:  L ← L(Y_sleep);
8:  for k in range(0, k_max) do
9:      Y'_sleep ← neighbor(Y_sleep);
10:     E' ← E(Y'_sleep);
11:     L' ← L(Y'_sleep);
12:     if E' > E and L' < L then
13:         Y_sleep ← Y'_sleep ;
14:     else if E' > E and L' ≥ L and L' < 𝕃 then
15:         P ← e^(L−L'/L) ;
16:         if randFloat(0, 1) > P then
17:             Y_sleep ← Y'_sleep ;
18:         end if
19:     else if E' < E and L' < L then
20:         P ← e^(E'−E/E') ;
21:         if randFloat(0, 1) > P then
22:             Y_sleep ← Y'_sleep ;
23:         end if
24:     end if
25: end for
```

FIGURE 5. Our simulated annealing-based algorithm deploys instances in the sleep layer.

finds Z_i for cloud i, whereas Subsubpack finds the value in log $(B_i - X_i - Y_i)$ time. When the cost or response latency exceeds the budget constraint or the maximum response latency with the capacity of cloud i, the algorithm uses the Subpack function. Otherwise, the algorithm invokes the Subsubpack function. Generally, because there are three loops in this algorithm, the time complex is $O(|C|\ \mathbb{D}\mathbb{L}\log B)$. Although this algorithm (Figure 6) seems more complicated than previous algorithms, it's acceptable because of its accuracy and the limited number of instances. To optimize the algorithm, we also set the latency unit to seconds and the budget to \$100. Therefore, the time overhead is

$$O\left(|C|\left\lfloor\frac{\mathbb{D}}{100}\right\rfloor\lfloor\mathbb{L}\rfloor\log B\right).$$

Performance Evaluation

We conducted simulation-based experiments to evaluate the proposed algorithms' performance. To evaluate performance in the general case, we generated random networks and compared the cost of different controller assignment algorithms.

For implementation and evaluation of the instance placement, we used the script language Python 2.7 and the NumPy library, and tested multiple workloads. We used price data from 24 cloud platforms and tested the status transition latency from snapshot to active status and the latency from active to service status. Because getting real-world concurrency data is difficult, we distributed the wakeup concurrency of each cloud platform evenly in range [1,000, 2,000] and the startup concurrency in range [200, 700]. Based on existing research,[5] we distributed each instance's service capacity evenly in range [100, 200] links. Meanwhile, we set the emergency response latency to 300 seconds, which is enough for the evacuation based on the existing analysis work[10] on the disaster.

For comparison, we considered three schemes:

- single cloud with dynamic programming,
- random deployment with a multicloud framework, and
- random deployment with a single cloud platform.

We first study the maximum service capacity in an emergency with different budgets for service maintenance. We set the budget to \$1,000 to \$5,000 per month. As Figure 4a shows, with a \$1,000 budget, maintaining a large enough cluster for the evacuation service in an emergency is difficult. However, with a larger budget, service capacity in an emergency increases rapidly. The ISCD deployment performs best in all solutions. Because the latency requirement isn't strict, the deployment on a single cloud performs at 80 percent of the capacity of the performance in MCES with the ISCD deployment. The performance of the two random deployment strategies is much lower than the dynamic algorithm and the ISCD deployment, especially with a large budget.

Next, we analyze the performance with strict response latency in an emergency. We tested the service capacity for a response latency of 50 to 250 seconds and a budget of \$2,000. With limited response latency, we need 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 than dynamic programming in the single cloud platform. With limited concurrency, the dynamic

programing performs a little better than the random deployment in the single cloud platform.

With the initial evaluation of our solution, using the same budget for service maintenance, we find that our platform can provide better service capacity than the single cloud platform in an emergency. MCES can maintain fewer active instances in periods of normal activity than a single cloud system, and use more snapshot and sleep instances, which have a lower cost. When a disaster occurs, with the higher concurrency provided by multiple cloud platforms, MCES can start up more instances than a single cloud platform with the given response latency (Figure 7b).

We also used a simulation based on a real-world scenario to evaluate whether the MCES provides capacity for an evacuation in a real disaster. We used data from the Indonesian city of Padang, which faces a high risk of being inundated by a tsunami. The city has more than 1 million people, of which 300,000 live in at-risk areas. Previous work has shown that people in these areas need more than 20 minutes to evacuate to a safe area, and the warning time before a tsunami reaches the coastline is only 20 to 40 minutes.[10] It's more strict than the simulation setting with the limited tsunami warning time. We use the same setting as the input to the simulation. Given that random deployment didn't perform well enough in the previous simulation, we only test the ISCD in the MCES and the dynamic programming-based placement in a single cloud platform.

We analyze the budget with different response latency requirements for the evacuation service of the Padang scenario. From the results, shown in Figure 7, we find that if the MCES costs less than $7,000 per month, the evacuation service response time is less than 1 minute, which is much less than the evacuation time when the single cloud-based evacuation service costs almost $11,000. If the response time request becomes less strict, the budget needed by the two systems decreases rapidly. When the response latency request is more than 4 minutes, the MCES requires less than $3,000, whereas the single cloud-based service requires less than $7,000. As a result, with the multicloud-based deployment support, MCES costs much less than the single cloud-based service, even in a real-world scenario.

```
1:  𝔻′ ← 𝔻 − Σ_{i=1}^{|C|} (X_i D_ri + Y_i D_si);
2:  𝕃′ ← 𝕃 − Σ_{i=1}^{|C|} Y_i N_si;
3:  for i from 1 to |C| do
4:      SnapshotPack(D_di, S_i, N_si B_i − X_i − Y_i);
5:      c, t ← arg max f;
6:      f[c] [0] ← f[c] [t];
7:  end for
8:  function SnapshotPack(d, s, n, b)
9:      if db >= 𝔻 and nb >= 𝕃′ then
10:         Subpack(d, s, n);
11:         return;
12:     end if
13:     k = 1;
14:     while k < b do
15:         Subsubpack(kc, ks, kn);
16:         b = b − k;
17:         k ← 2k;
18:     end while
19: end function
20: function Subpack(d, s, n)
21:     for d′ from d to 𝔻′ do
22:         for n′ from n to 𝕃′ do
23:             f[d′][n′] ← max(f[d′][n′], f[d′ − D_di][n′ − N_di] + S_i);
24:         end for
25:     end for
26: end function
27: function Subsubpack(d, s, n);
28:     for d′ from c to 𝔻′ do
29:         for n′ from n to 𝕃′ do
30:             f[d′][n′] ← max(f[d′][n′], f[d′ − d][n′ − n] + s);
31:         end for
32:     end for
33: end function
```

FIGURE 6. The dynamic programming-based algorithm for deploying instances in the snapshot layer.

We've shown how MDPS could potentially solve the current limitations of single cloud-based evacuation systems in an emergency by deploying services to multiple cloud platforms. In the future, we plan to implement a complete MCES framework as a module in OpenStack.[11] We also need to further experiment with the real-world testbed to evaluate the efficiency of the MCES framework.

Acknowledgments

This work is partially supported by the Japan Society for the Promotion of Sciences (JSPS) KAKENHI grant numbers 25880002, 26730056; JSPS A3 Foresight Program; National Natural Science Foundation of China (NSFC) grant numbers 61450110085, 61272444, 61411146001, U1401253, and U1405251.

References

1. R.N. Calheiros et al., "Cloudsim: A Toolkit for

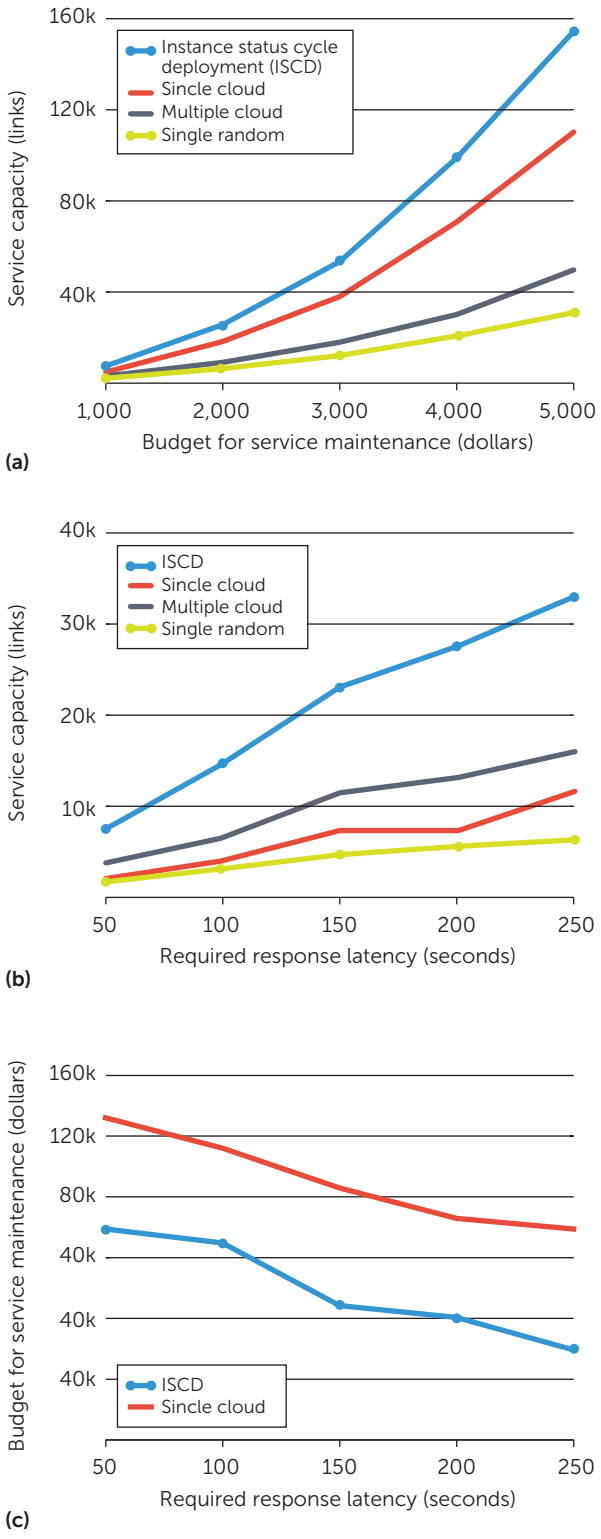

FIGURE 7. Performance evaluations results: (a) service capacity with different budgets for service maintenance; (b) service capacity with different required response latency; and (c) minimum budget for the evacuation service in the Padang scenario.

Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and Experience*, vol. 41, no. 1, 2011, pp. 23–50.

2. M. Armbrust et al., "A View of Cloud Computing," *Comm. ACM*, vol. 53, no. 4, 2010, pp. 50–58.
3. M. Dong et al., "HVSTO: Efficient Privacy Preserving Hybrid Storage in Cloud Data Center," *Proc. IEEE Conf. Computer Comm. Workshops* (INFOCOM 14), 2014, pp.529, 534; doi: 10.1109/INFCOMW.2014.6849287.
4. Z. Alazawi et al., "Intelligent Disaster Management System Based on Cloud-Enabled Vehicular Networks," *Proc. 11th Int'l Conf. ITS Telecomm.* (ITST 11), 2011, pp. 361–368.
5. L. Chu and S.-J. Wu, "An Integrated Building Fire Evacuation System with RFID and Cloud Computing," *Proc. 7th Int'l Conf. Intelligent Information Hiding and Multimedia Signal Processing* (IIH-MSP 11), 2011, pp. 17–20.
6. X. Pu et el., "Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments," *Proc. IEEE 3rd Int'l Conf. Cloud Computing* (Cloud 10), 2010, pp. 51–58.
7. I. Houidi et al., "Cloud Service Delivery across Multiple Cloud Platforms," *Proc. IEEE Int'l Conf. Services Computing* (SCC 11), 2011, pp. 741–742.
8. M. Dou et al., "Modeling and Simulation for Natural Disaster Contingency Planning Driven by High-Resolution Remote Sensing Images," *Future Generation Computer Systems*, vol. 37, no. 37, 2014, pp. 367 – 377.
9. S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal Virtual Machine Placement across Multiple Cloud Providers," *Proc. IEEE Asia-Pacific Services Computing Conf.* (APSCC 09), 2009, pp. 103–110.
10. G. Lämmel et al., "Emergency Preparedness in the Case of a Tsunami: Evacuation Analysis and Traffic Optimization for the Indonesian City of Padang," *Pedestrian and Evacuation Dynamics 2008*, W.W.F. Klingsch et al., eds., Springer, 2010, pp. 171–182.
11. K. Pepple, *Deploying OpenStack*, O'Reilly Media, 2011.

MIANXIONG DONG *is an assistant professor in the Department of Information and Electronic Engineering, Muroran Institute of Technology, Japan, and a research scientist with the A3 Foresight Program, funded by the 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. Dong has a PhD in computer science and engineering from the University of Aizu, Japan. Contact him at mx.dong@csse.muroran-it.ac.jp.

HE LI *is a PhD student in discipline at the Graduate School of Computer Science and Engineering, Huazhong University of Science and Technology, China. His research interests include cloud computing and software-defined networking. Li has an MS in discipline from Huazhong University of Science and Technology. He is a student member of IEEE and the IEEE Communication Society. Contact him at heli @hust.edu.cn.*

KAORU OTA *is an assistant professor with the Department of Information and Electronic Engineering, Muroran Institute of Technology, Japan. Her research interests include wireless sensor networks, vehicular ad hoc networks, and ubiquitous computing. Ota has a PhD in computer science and engineering from the University of Aizu, Japan. Contact her at ota@csse .muroran-it.ac.jp.*

LAURENCE T. YANG *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, and big data. His research has been supported by the National Sciences and Engineering Research Council, and the Canada Foundation for Innovation. Yang has a PhD in computer science from the University of Victoria, Canada. Contact him at ltyang@gmail.com.*

HAOJIN ZHU *is an associate professor in the 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. Zhu has a PhD in electrical and computer engineering from the University of Waterloo, Canada. Contact him at zhu-hj@cs.sjtu.edu.cn.*

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.