Dynamic Resource Management in Cloud-based Distributed Virtual Environments

Yunhua Deng[†], Siqi Shen[‡], Zhe Huang[†], Alexandru Iosup[‡], Rynson Lau[†]
[†]City University of Hong Kong, China [‡]Delft University of Technology, Netherlands
yunhua.deng@my.cityu.edu.hk, s.shen@tudelft.nl, igamenovoer@gmail.com
a.iosup@tudelft.nl, rynson.lau@cityu.edu.hk

ABSTRACT

As an elastic hosting platform, cloud computing has been attracting many attentions for transferring compute-intensive applications from static self-hosting to flexible cloud-based hosting. Distributed virtual environments (DVEs) which typically involve massive users interacting at the same time and feature significant workload dynamics either in spatial due to in-game user mobility or in temporal due to the fluctuating user population, potentially are suitable applications with cloud-based hosting because of the need of resource elasticity. We explore the dynamic resource management for cloud-based DVEs by taking into account their multi-level workload dynamics which differ them from other applications. Simulation results demonstrates the advantages of our developed methods over existing ones.

Categories and Subject Descriptors

Information systems [**Information systems applications**]: Multimedia information systems; Computer systems organization [**Distributed architectures**]: Cloud computing

Keywords

Distributed Virtual Environments; Virtual Worlds; Resource Management; Load Balancing; Cloud Computing; Elasticity

1. INTRODUCTION

Distributed (or networked) virtual environments (DVEs or NVEs) are systems in which users interact with each other and with virtual objects in a shared digital space in real-time through Internet. DVEs are increasingly popular and currently represent a range of applications, such as massively multiplayer online games (MMOGs) which have a specific purpose for gaming (e.g., World of Warcraft) and virtual worlds which have general purposes like collaborative learning, training and socializing (e.g., Second Life). DVEs are designed to create very high degree of immersion with realistic 3D graphics [7] and stereo sound [10] and to support massive users to co-exist in a single and consistent virtual space with scalable server architectures. Most existing MMOGs commonly use the sharding approach by creating multiple copies of the game world each of which can be handled by a server and forcing users to choose to play in one of those independent copies. As users are only allowed to play and interact with others within the same copy hosted by the same server, the scalability problem can be avoided by explicitly limiting the number of users in each copy. Limitations for this sharding approach include not being able to interact with every user and most severely the loss of shared experiences perceived by users. As pointed out by [4], this sharding approach makes the current MMOGs different from general seamless DVEs, as that must maintain a single, persistent and consistent state.

Many research have been made to improve DVE scalability to support larger numbers of users, and most of them are based on distributed-server architectures. On one hand, some efforts focus on reducing the inter-server communication traffic induced by the consistency maintenance using their proposed message filtering techniques. On the other hand, some efforts focus on dynamically adjusting the workload-to-server assignment to adapt to changes in workload distribution among servers to avoid server overloading based on their proposed dynamic load balancing techniques.

Like many Internet-based applications, DVEs tend to feature significant workload fluctuations in different time scales (e.g., hourly, diurnally, weekly, or monthly). This poses another issue apart from the scalability issue and we call it *elasticity issue* which represents the need to adjust server resources dynamically to adapt to global workload changes for achieving high resource efficiency. Previous approaches assume the total load of the system is static; this assumption is not true due to the fluctuation in the number of online players. Their approaches need to provision the servers according to the peak load which leads to waste of resources. Their dynamic load balancing techniques can prevent individual servers from being overloaded only when the global workload remains unchanged or becomes less. If the global workload surpasses the total capacity provided by the currently deployed servers, individual server overloading cannot be avoided anyway. On the other hand, if the global workload reduces significantly, it is not efficient to keep the same amount of capacity as in the peak periods.

Bewaring of the global workload fluctuation nature in MMOGs, a few recent work have been proposed to bring cloud computing paradigm for efficiently hosting MMOGs. For instance, Nae et al. [8] develop a predictor based on neural networks to forecast the global workload and request properly sufficient resources from external data centers (cloud providers) where MMOGs are hosted. Their study is based on a MMOG called RuneScape that uses sharding (i.e., users cannot move across different servers). Hence, as long as prediction for each server's workload is accurate, there will be no servers getting overloaded. However, in general DVEs that do not use sharding, prediction of the workload in each server is much more difficult as users can move freely instead of keeping staying in the copy hosted by a specific server. For another example, Lee and Chan [6] propose to apply server consolidation technique to adaptively change the number of servers according to the workload changes via periodically consolidating regions of the game world into nearly fewest number of servers based on a bin packing heuristic (first-fit-decreasing). As long as the workload distributed among each region remains relatively constant which is basically true for sharded MMOGs (i.e., users are not moving across different regions handled by different servers) between two consecutive consolidation processes, their approach can work appropriately without causing server overloading. However, the works discussed above are targeted to sharded MMOGs and may not be appropriate for contiguous DVEs [4].

In this work, we focus on the dynamic resource management problem for seamless DVEs. Specifically, we consider both global workload changes induced by user population fluctuations and local workload changes induced by user mobility. Our proposed approach not only improves resource efficiency as compared to those which overlook global workload dynamics (e.g., [3]), but also reduces service interruption induced by resource reallocation as compared to those which overlook local workload dynamics (e.g., [6]). We present the proposed approach and simulation results in Sections 2 and 3, respectively.

2. PROPOSED APPROACH

To support a large number of users co-existing in a shared virtual world, we adopt multi-server DVE architecture with each server handling a partition of the virtual world (and users therein) and together all servers maintaining a single and shared virtual world. For the ease of management and extension, usually the virtual world is designed to be composed of a set of regular-shaped regions (e.g., Second Life consists of over 7,000 regions which form a contiguous map), and each server is assigned a subset of them. Each server maintains the responsiveness and consistency perceived by users residing in the regions within its management.

2.1 Problem Statement

There are two issues induced by user mobility across different regions. The first issue is handover, which happens when a user travels from its current region to a new one (normally it is adjacent but sometimes can be remote if teleporting). An inter-server handover is caused if the new region is hosted by another server which apparently should be more difficult to be handled transparently compared to the intra-server handover (i.e., the new region is hosted by the same server as the old one). Hence, one should consider assigning adjacent regions to the same servers as much as possible. The second issue is that the user distribution across regions is not fixed but dynamic. Hence, the originally load balanced region-to-server assignment may become imbalanced and occasionally some servers may get heavily loaded such that they cannot service more coming users (i.e., *locally under-provisioned*). As long as the total server capacity is enough to host all the users, we can reduce the workload assigned to those under-provisioned servers by transferring some regions from them to those which still have enough capacity.

However, if the total number of users grows significantly such that the total capacity of all servers becomes insufficient (i.e., *globally under-provisioned*), any load balancing process is useless and inevitably we must either add new servers or increase the capacity of existing servers.. The former strategy is generally referred to as *scaling out* while the latter one is *scaling-up*. On the other hand, it should be beneficial if we can reduce the capacity provision during low load periods (i.e., *globally over-provisioned*) to achieve resource efficiency. Strategies related to decreasing the number of servers and existing servers' capacities are referred to as *scaling in* and *scaling down*, respectively. Note that in cloud computing commonly the scaling out or in process is called *horizontal scaling*, while the scaling up or down process is called *vertical scaling*. Figure 1 depicts the framework of our dynamic resource management approach.

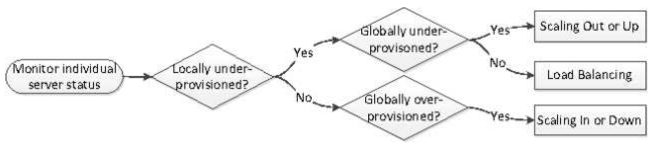
Monitor individual server status
Locally under-provisioned?
Yes
No
Globally under-provisioned?
Globally over-provisioned?
Yes
No
Yes
Scaling Out or Up
Load Balancing
Scaling In or Down

Figure 1: DVE dynamic resource management.

Let's denote $\{s_1, s_2, ..., s_n\}$ as the set of n servers, with $\{l_1, l_2, ..., l_n\}$ and $\{c_1, c_2, ..., c_n\}$ representing their corresponding loads and capacities, respectively. $l^{ttl} = \sum_1^n l_i$ and $c^{ttl} = \sum_1^n c_i$ are the total capacity and total load respectively. As shown by the framework in Figure 1, the resource management process will encounter one of these circumstances: locally under-provisioned if there is any s_i has $c_i \leq l_i * (1 + \sigma)$, with σ denoting the minimum required resource redundancy of each server; globally under-provisioned if $c^{ttl} \leq l^{ttl} * (1 + \theta)$, with θ denoting the desired global resource redundancy of all servers ($\theta > \sigma$); globally over-provisioned if $c^{ttl} \geq l^{ttl} * (1 + \delta)$, with δ denoting the maximum allowed resource redundancy of all servers ($\delta > \theta$).

2.2 Horizontal Scaling

Once detecting $c_i \leq l_i * (1 + \sigma)$ and $c^{ttl} \leq l^{ttl} * (1 + \theta)$, we trigger *scaling out* algorithm described as follows:

1. add a new server (i.e., request a new instance from the cloud provider), namely s_{n+1}, and transfer regions from the most loaded server, namely s_k, to s_{n+1} until $c_k \geq l_k * (1 + \theta)$ or $c_{n+1} \leq l_{n+1} * (1 + \theta)$;

2. if either $c_i \leq l_i * (1 + \sigma)$ or $c^{ttl} \leq l^{ttl} * (1 + \theta)$, go to step 1; otherwise, terminate.

After termination of the above procedure, each server s_i finally has its capacity $c_i \geq l_i * (1+\theta)$ such that achieving the desired global resource redundancy, i.e., $c^{ttl} \geq l^{ttl} * (1 + \theta)$.

Once detecting that none of the servers is locally under-provisioned and $c^{ttl} \geq l^{ttl} * (1 + \delta)$, we trigger *scaling in* algorithm described as follows:

1. merge s_i with its neighbor s_j (two servers are considered as neighboring if they host adjacent regions) if $c_j \geq (l_i + l_j) * (1 + \theta)$ and $(c^{ttl} - c_i) \geq l^{ttl} * (1 + \theta)$ by transferring all regions from s_i to s_j and release s_i;

2. repeat step 1 until having gone through all its neighbors; terminate if having gone through all servers.

Note that δ should be set significantly larger than θ in order to avoid unnecessary oscillations between scaling in and out.

The major advantage of horizontal scaling is that one can adjust the capacity to adapt to any workload changes. However, the major drawback is the inevitable inter-server region migrations induced by scaling out or in actions which may affect user experiences and may be worse than the similar migration issue encountered by load balancing among servers because of launching new server instances could take some time when scaling out. Hence, we consider another way for dynamic resource management.

2.3 Vertical Scaling

Normally vertical scaling is done through replacing the old servers by new ones having different capacities [9]. Indeed, the replacement process itself could also affect user experiences as the horizontal scaling does. Recently, there are some advances in both academia (e.g., [2, 5]) and IT industry (e.g., ProfitBricks) making live vertical scaling viable, that is, adjusting servers' sizes in an on-the-fly manner. We make such an assumption for our vertical scaling algorithms.

With the same triggering condition as scaling out algorithm, *scaling up* algorithm can be described as follows:

1. enlarge the capacity of s_i which has $c_i \leq l_i * (1+\sigma)$ until $c_i \geq l_i * (1+\theta)$ or $c_i = c^{max}$ (i.e., the maximum allowed size of a server); if $c^{ttl} \geq l^{ttl}*(1+\theta)$, terminate; otherwise, go to next step;

2. enlarge the smallest server s_k (i.e., having the lowest capacity) until $c^{ttl} \geq l^{ttl}*(1+\theta)$ or $c_k = c^{max}$; if $c^{ttl} < l^{ttl} * (1+\theta)$, repeat this step; otherwise, terminate.

Having the same triggering condition as scaling in algorithm, *scaling down* algorithm can be described as follows:

1. decrease the largest server s_k (i.e., having the highest capacity) until $c_k \leq l_k * (1+\theta)$, $c_k = c^{min}$ (i.e., the minimum allowed size of a server), or $c^{ttl} \leq l^{ttl}*(1+\theta)$;

2. repeat step 1 until having gone through all servers.

Note that after vertical scaling, servers that are homogeneous originally may become heterogeneous, i.e., having different capacities. This should be taken into account in load balancing, that is, the workload assigned to each server should be proportional to its capacity. However, most existing DVE dynamic load balancing techniques (such as [3] and references therein) do assume that servers are homogeneous. Hence, we extend the method proposed by [3] to handle the server heterogeneity when doing load balancing, i.e., the time when locally under-provisioned event arises but globally under-provisioned event doesn't as illustrated in Figure 1. So far, we have presented our proposed resource management approach and the associated algorithms. Next we present experiments for performance evaluation.

3. EXPERIMENTAL RESULTS

3.1 Simulation Settings

We simulate a virtual world consisting of 10,000 regions each of which has an area of $256m \times 256m$ (comparable to Second Life). We describe the simulation for user mobility as follows. Each user stays in his current region for a period of time randomly sampled from the range of $[T_{min}, T_{max}]$. Upon spending up the chosen time period, he will move to a new region randomly chosen from the adjacent regions of his current region or a random remote region (in case for teleporting). The remote region one of the popular regions, called hotspots, which drives uneven user distributions and we set 20% of regions as hotspots (similar to [3]). Generally, users are moving faster and more frequently for exploration or questing than social interaction. Hence, we simulate two scenarios: *slow-paced mobility*, with T_{min} = 10 minutes and T_{max} = 30 minutes; *fast-pace mobility*, with T_{min} = 1 minute and T_{max} = 10 minutes. For user population fluctuations, we adopt the trace used in [6] and magnify it to 100,000 users at the peak period. The simulated period is 1 day with 1 minute per timestep.

Without loss of generality, we assume the maximum number of users can be supported by a server is proportional to its CPU cores. If using instances in Amazon EC2, we can have servers with sizes ranging from 2 to 32 CPU cores (note that the number is the powers of 2). One core can handle up to 50 concurrent users in to Second Life, meaning that one server can support up to 1,600 users. Except for virtual scaling, we assume all servers have the maximum allowed capacity by default.

We compare these methods: *BP*, representing the bin packing based consolidation method [6]; *NS*, representing the method of dynamic load balancing with no scaling [3]; *HS*, representing our horizontal scaling based method; *VS*, representing our vertical scaling based method.

We have the following metrics: *resource allocation*, measured by the number of CPU cores allocated by all servers; *service interruption count*, measured by the number of non-serviced users as they enter the regions managed by fully loaded servers which cannot service any new coming users; *handover count*, measured by the number of inter-server user handovers; *migration count*, measured by the number of users inside those regions being migrated.

The parameters defined in Section 2.1, i.e., σ, θ, and δ are set as follows. $\sigma = 10\%$, meaning that the locally under-provisioned event will be issued if any server s_i has $c_i \leq (100+10)\% * l_i$. $\delta = 200\%$, meaning that the globally over-provisioned event will be issued if $c^{ttl} \geq (100+200)\% * l^{ttl}$. For θ, we have three options: 25%, 50%, and 100%. Higher θ reduces the chance of server overloading but increases the resource allocation, and vice versa.

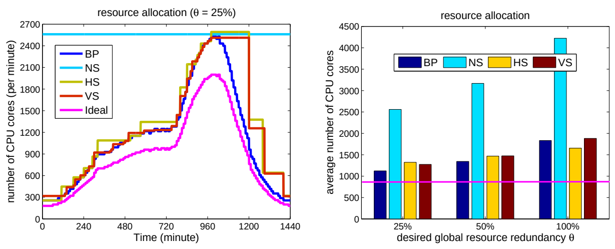

Figure 2: Comparison on instantaneous (left) and average (right) resource allocation.

3.2 Results and Analysis

As results taken with both mobility scenarios are similar, we present only those with fast-paced scenario. Figure 2 (left) shows the resource allocation by each method. We can see that *BP*, *HS* and *VS* adaptively adjusts the resource according to the actual demand (referring to the pink curve), while *NS* statically allocates the amount of resource for preparing the peak load period (around the 1000*th* minute). The average values are shown in Figure 2 (right), and we can see that the allocated resource increases with θ for each method. For example, when $\theta = 100\%$, *BP*, *HS* and *VS* allocate nearly twice as the actual demanded resource on average. Overall, our *HS* and *VS* methods improve the resource efficiency compared to *NS*. Next we explore the advantages of our methods over *BP*.

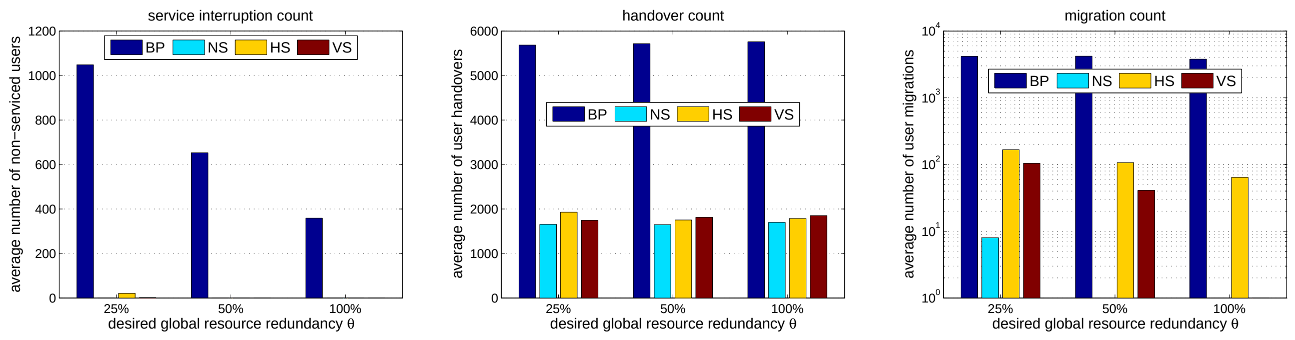

Figure 3: Comparison on interruption count (left), handover count (middle), and migration count (right).

Figure 3 (left) shows that *BP* causes lots of non-serviced users and increasing θ will decrease the service interruption. In contrast, our *HS* and *VS* roughly produces no service interruption as *NS* does even for a small θ, which implicitly indicates that *HS* and *VS* can achieve better resource efficiency than *BP* by using small θ while still maintaining the service but *BP* cannot.

Figure 3 (middle) shows the average amount of user handovers per minute. *BP* has nearly two times of users handovers more than others. This is primarily because *BP* method overlooks the adjacency of regions when assigning them to servers. Apparently there will be more users moving across servers if the adjacent regions of their currently regions are not in the same servers, resulting more frequent changes of workload distribution among servers and then increasing the chance for a originally normally loaded server to quickly become fully loaded.

In Figure 3 (right), we can see that *BP* causes much higher migration count than others. The reason are two-fold. On one hand, as servers are frequently to get fully loaded, *BP* method triggers the resource adjustment process frequently by partitioning the regions into more servers and apparently this will induce many user migrations. On the other hand, repartitioning regions to servers, it does not take the current region-to-server mapping into account, such that regions are likely to be assigned to servers different from their original ones. When comparing between *HS* and *VS*, the latter evidently causes less user migrations, confirming that vertical scaling is potentially a more efficient way than horizontal scaling for cloud-based DVEs. Overall, our methods *HS* and *VS* outperform *BP* by having much smaller impact on system performance while maintaining resource efficiency.

4. CONCLUSION

In this paper, we explore dynamic resource management for seamless DVEs with the objective of improving resource efficiency while not affecting system performance. In specific, we develop horizontal and vertical scaling methods with the incorporation of our recently developed DVE load balancing approach. Simulation results show that our methods significantly outperform existing ones in general.

As an initial work in this field, this paper has several unanswered questions. For instance, we need to compare the horizontal scaling and vertical scaling for DVE dynamic resource management by taking into account all system overheads. Also, we need to conduct the evaluation using real-world deployment and realistic workload traces. Most importantly, we need to evaluate whether the latency perceived by users when we host DVEs in the cloud with dynamic resource management can still be acceptable compared to the time when we host DVEs with dedicated and static server allocation (similar to the study by Barker and Shenoy [1]).

5. ACKNOWLEDGMENTS

This work was partially supported by a GRF grant from RGC of Hong Kong (Project No.: CityU 115112), by the National Basic Research Program of China (2011CB302603 and 2014CB340303), and by the STW/NOW veni grant 11881.

6. REFERENCES

[1] S. Barker and P. Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proc. ACM Multimedia Systems*, pages 35–46, 2010.

[2] O. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. The resource-as-a-service (raas) cloud. In *Proc. USENIX HotCloud*, 2012.

[3] Y. Deng and R. Lau. Dynamic load balancing in distributed virtual environments using heat diffusion. *ACM TOMCCAP*, 10(2), 2014.

[4] J. Dionisio, W. Burns, and R. Gilbert. 3d virtual worlds and the metaverse: Current status and future possibilities. *ACM Computing Surveys*, 45(3), 2013.

[5] R. Han, L. Guo, M. Ghanem, and Y. Guo. Lightweight resource scaling for cloud applications. In *Proc. IEEE/ACM CCGrid*, pages 644–651, 2012.

[6] Y. Lee and K. Chen. Is server consolidation beneficial to mmorpg? a case study of world of warcraft. In *IEEE CLOUD*, pages 435–442, 2010.

[7] S. Mondet, W. Cheng, G. Morin, R. Grigoras, F. Boudon, and W. Ooi. Streaming of plants in distributed virtual environments. In *Proc. ACM Multimedia*, 2008.

[8] V. Nae, A. Iosup, and R. Prodan. Dynamic resource provisioning in massively multiplayer online games. *IEEE TPDS*, 22(3):380–395, 2011.

[9] L. Vaquero, L. Rodero-Merino, and R. Buyya. Dynamically scaling applications in the cloud. *ACM Computer Communication Review*, 41(1):45–52, 2011.

[10] R. Zimmermann and K. Liang. Spatialized audio streaming for networked virtual environments. In *Proc. ACM Multimedia*, pages 299–308, 2008.