Deadline-Aware Deployment for Time Critical Applications in Clouds

Yang Hu[1,2(✉)], Junchao Wang[1], Huan Zhou[1,2], Paul Martin[1], Arie Taal[1], Cees de Laat[1], and Zhiming Zhao[1]

[1] University of Amsterdam, Amsterdam, The Netherlands
{Y.Hu,j.wang2,H.Zhou,P.W.Martin,A.Taal,delaat,Z.Zhao}@uva.nl
[2] National University of Defense Technology, Changsha, China

Abstract. Time critical applications are appealing to deploy in clouds due to the elasticity of cloud resources and their on-demand nature. However, support for deploying application components with strict deadlines on their deployment is lacking in current cloud providers. This is particularly important for adaptive applications that must automatically and seamlessly scale, migrate, or recover swiftly from failures. A common deployment procedure is to transmit application packages from the application provider to the cloud, and install the application there. Thus, users need to manually deploy their applications into clouds step by step with no guarantee regarding deadlines. In this work, we propose a Deadline-aware Deployment System (DDS) for time critical applications in clouds. DDS enables users to automatically deploy applications into clouds. We design bandwidth-aware EDF scheduling algorithms in DDS that minimize the number of deployments that miss their deadlines and maximize the utilization of network bandwidth. In the evaluation, we show that DDS leverages network bandwidth sufficiently, and significantly reduces the number of missed deadlines during deployment.

1 Introduction

Cloud computing is the platform of choice for deploying and running many of today's businesses. When executing applications in clouds, deployment is an important step to make required software and data of an application available before execution. In cloud environments, Software as a Service (SaaS), e.g., Google Apps, or Platform as a Service (PaaS), e.g., Amazon EMR, aim at hiding the deployment complexity by automating deployment during resource provisioning [13]. However, these solutions are not sufficient for applications that require infrastructure-level optimization under the given platform services or application-level customized environments, which are not included in predefined virtual machines or container images.

Time critical applications, such as disaster early warning systems, often have very high performance requirements for data communication and processing [18]. To support time critical applications using cloud environments, developers often use Infrastructure as a Service (IaaS) to optimize overall system-level performance by selecting the most suitable virtual machines, customizing their network

F.F. Rivera et al. (Eds.): Euro-Par 2017, LNCS 10417, pp. 345–357, 2017.
DOI: 10.1007/978-3-319-64203-1_25

topology and optimizing the scheduling of execution on the virtual infrastructure [6,16,20]. During the execution, the virtual infrastructure often has to be adapted, e.g., virtual machines scaling out/in or up/down to handle dynamically changing workloads [19]. A deployment service is thus needed not only before the application execution for making the environment available, but also at runtime. In particular, it is necessary to ensure that components can be deployed immediately whenever the application needs to re- scale to handle increased workloads, or migrate components to new VMs. Moving the repository of components closer to the application is necessary to ensure that such deployments can be handled as rapidly as possible for time critical applications. Furthermore, the deployment service also has to be aware of time constraints, e.g., deadlines, required for acceptable system performance. Deployments that fail to finish within certain deadlines harm user experience, affect application performance, and even incur penalties for application failure. However current cloud providers lack explicit support for deploying time critical applications where users need to manually deploy their applications step by step and have no guarantee regarding deployment deadlines.

In this paper, we propose a Deadline-aware Deployment System (DDS) for time critical applications in clouds. DDS enables users to automatically deploy time critical applications and provide scheduling mechanisms to guarantee deployment deadlines. First, DDS helps users to create a local repository for application components instead of using a remote repository, providing a guarantee of bandwidth for transmitting application packages where the transmission rate directly from the remote repository is widely varying. To be deadline-aware, DDS schedules deployment requests based on Earliest Deadline First (EDF) [8] which is a classical scheduling technique to minimize the number of deployments that miss deadlines. Furthermore, we design bandwidth-aware EDF to facilitate DDS to satisfy a greater number of deadline requirements and achieve sufficient utilization of bandwidth. In the evaluation, we demonstrate that DDS significantly reduces the number of deployments that miss deadlines, and leverages bandwidth sufficiently. We summarize our contributions as follows:

- We design and implement DDS, a deadline-aware deployment system which can support automatic deployments of time critical applications in clouds.
- We build on DDS to implement deployment scheduling algorithms that minimize the number of deployments that miss deadlines and maximize the utilization of bandwidth.
- We experimentally evaluate the benefits of DDS on the ExoGENI [2] test-bed and large-scale simulations by comparing it with three different scheduling techniques.

2 Problem Statement

A typical scenario for deploying distributed applications in clouds involves two basic steps: transmitting necessary application packages or software components

from remote repositories to virtual machines (VMs) in the provisioned infrastructure; and installing the software once runnable. In this paper, we assume containers, e.g., Docker [9], are the default way to wrap application components.

For a distributed application, the deployment service has to know the location of application components, and the location to deploy (VMs) for each component. Those container images are often stored in a repository, e.g., Docker hub, which is not a part of the provisioned virtual infrastructure. The deployment service should schedule the sequence of each component based on the application description for transmitting and installing each individual component. The time for deploying a single container (T_d) typically contains time cost for transmitting the component from its repository (T_f) and installing (extracting files from the Docker image) the component (T_i). The total time of the deployment of the whole application starts from the first component transmission until the last component finishes its installation. When an application contains more components, careless scheduling of the deployment sequence might lead to a high time cost, which can eventually influence the execution of the application if key application components are delayed during deployment.

T_f depends on the size of the container and the network bandwidth between repository and target. T_i mainly depends on the performance of the VM and the complexity of the container itself. In many cases, T_f is much bigger than T_i. Table 1 shows some observations in a private cloud environment (ExoGENI [2]). We created VMs which are "xo.medium" configuration in three different locations: Boston, Washington and Houston. We found that T_f is widely varying because the internet connection between VMs and Docker hub is different between different locations, and T_i is stable for the same VM configurations. For meeting the deployment time constraints of time critical applications in provisioned virtual infrastructure, the key challenge is how to minimize the transmission time T_f and predict the installation time T_i. Installation time prediction is not the focus on this paper—we assume that existing predictors [11] can achieve good estimations of installation time. In this paper, we focus on the transmission process (T_f) of deployment.

Table 1. Comparison of transmission time and installation time in different locations

Docker image	Image size	Boston rack	Washington rack	Houston rack
ubuntu	400 Mb	T_f : 40.8 s(±2.2 s)	T_f : 27.0 s(±1.5 s)	T_f : 20.3 s(±1.5 s)
		T_i : 6.3 s(±0.5 s)	T_i : 6.4 s(±0.4 s)	T_i : 6.3 s(±0.6 s)
nginx	576 Mb	T_f : 58.7 s(±2.5 s)	T_f : 38.9 s(±2.6 s)	T_f : 29.2 s(±1.8 s)
		T_i : 9.3 s(±0.7 s)	T_i : 9.1 s(±0.5 s)	T_i : 9.3 s(±0.6 s)
mongodb	1200 Mb	T_f : 122.4 s(±3.0 s)	T_f : 81.0 s(±3.4 s)	T_f : 60.9 s(±1.9 s)
		T_i : 15.4 s(±0.5 s)	T_i : 15.7 s(±0.8 s)	T_i : 15.5 s(±0.8 s)
cassandra	1296 Mb	T_f : 132.2 s(±3.1 s)	T_f : 87.5 s(±3.4 s)	T_f : 65.7 s(±2.3 s)
		T_i : 17.1 s(±0.9 s)	T_i : 17.3 s(±0.7 s)	T_i : 17.4 s(±0.6 s)

The deployment model in this work is a set of deployment requests. The deployment service has to optimize the time cost by scheduling component transmissions carefully, and parallelize the data transfer based on the time constraint obtained from the application. We model the deployment request as a tuple $R_i = (v_i, s_i, d_i)$, where v_i is the target virtual machine to deploy request R_i, s_i is the application size (e.g., Mb), and d_i is its deadline. As we concentrate on transmission, we model bandwidth information for provisioned VMs as sets $B = \{b_1, b_2, b_3, \ldots, b_n\}$, where b_i denotes the bandwidth of virtual machine i. This means that the *throughput* of virtual machine i can not exceed b_i during the transmission process, and the bandwidth is stable based on the SLA provisioning mechanisms [3] in this context. We denote the bandwidth of the target machine v_i as b_j, so that the transmission time of request R_i can be represented as $T_f = \frac{s_i}{b_j}$. Similarly, the deployment time can be represented as $T_d = \frac{s_i}{b_j} + T_i$. The problem of this paper is thus to investigate the scheduling mechanisms needed to meet the deployment deadlines (i.e., ensure that $T_d \leq d_i$) of time critical applications in clouds.

3 Deadline-Aware Deployment System

This section highlights our approach in DDS. DDS aims to provide a deadline-aware, efficient and automatic deployment system that supports time critical applications on infrastructure as a service on cloud systems. As we mainly consider the transmission part of the deployment procedure in this paper, DDS focuses on the network of the underlying distributed system to provide the best guarantee for deployment within deadlines.

3.1 Design Principles

Repository Location. The repository for the application is a shared storage from which application packages can be fetched to be installed on another machine. The repository can be located in a remote server or in the cloud already. The location of the repository can directly impact the deployment time because the network bandwidth between cloud VMs and between a VM and a remote repository in a different location can be very different. Compared to a remote repository, a local repository within a cloud has some obvious advantages. First, the local repository has greater transmission capacity than the remote repository. Second, the bandwidth of the local repository inside a cloud is more stable, which provides a guarantee regarding the transmission time. Third, the local repository is more flexible due to the possibility of personalized configuration. Thus, DDS would help users to create a local repository first if there is only a remote repository from which to fetch application packages.

Deadline-Aware Mechanism. As the goal of DDS to meet the deadline of requests, whether the system is aware of the deadline is important for deployment. Consider a common time critical application scenario involving two deployment requests sent to the same application component provider simultaneously,

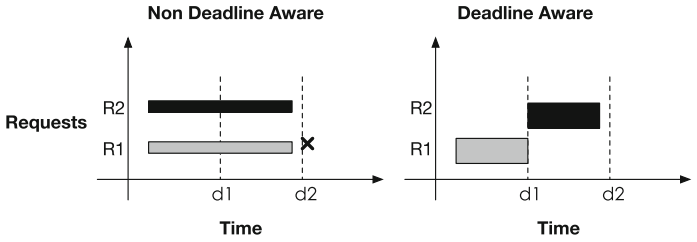

Fig. 1. Awareness of deadlines can be used to meet two deadlines

where one request has a tighter deadline than the other. The resulting requests share a bottleneck via which to transmit application packages. As shown in Fig. 1, with today's setup, the transport protocol (e.g., TCP) strives for fairness and the transmission finishes for both requests almost simultaneously. However, only one of the requests meets its deadline which makes the another request useless or degrades its value. Alternatively, given explicit information about deployment deadlines, the system can arrange the transmission order to better meet the deployment deadline.

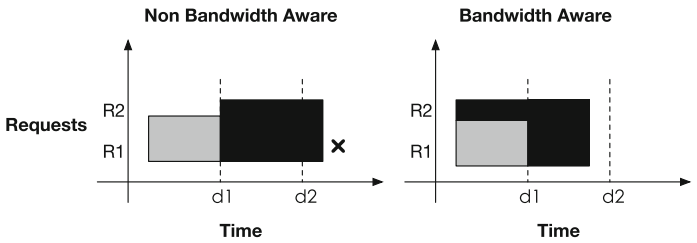

Fig. 2. Awareness of bandwidth can be used to meet two deadlines

Bandwidth-Aware Mechanism. In addition to deadline-aware scheduling, to be aware of bandwidth is another significant attribute for deployment. Consider another scenario with two deployment requests, where the second request pulls a larger application package. The resulting requests also share a link to transmit their respective packages. As shown in Fig. 2, the deployment system has information about the deadlines and schedules the transmission based on those deadlines. However only one request meets its deadline. Because the transmission bottleneck is the bandwidth of the target machine, there is some spare bandwidth on the server which is not used. Thus, given explicit information about the bandwidth capacity of each machine in the cloud, the system could schedule more deployment requests and leverage the bandwidth more efficiently.

3.2 Scheduling Algorithm

In this section, we zoom in on the design principles presented in Sect. 3.1 by providing an algorithmic description. The main goal of our algorithms is to minimize the deadline miss rate: the application packages should be transmitted to the target machine within the deadline wherever possible. In addition to minimizing miss rate, we should maximize the bandwidth utilization to reduce the total transmission time. To achieve both these goals, we employ EDF to prioritize requests and design bandwidth-aware EDF to support parallel transmission and realize dynamic rate control.

EDF Scheduling. The key insight guiding the design of deadline-aware scheduling is derived from the classic real-time scheduling algorithm Earliest Deadline First (EDF) [8], which prioritizes tasks based on their deadline. EDF is an optimal scheduling algorithm in that if a set of deadlines can be satisfied under some schedule, then EDF can satisfy them too.

We adopt EDF to schedule deployment requests. When a deployment request comes, DDS compares the deadline of new request with previous requests and then sets the corresponding priority relative to the other deadlines. DDS then puts the new request into the request queue where the requests are sorted by priority. The algorithm is described in Algorithm 1. Consequently, DDS obtains the request from the queue and starts to transmit application packages to the target machine.

Algorithm 1. EDF scheduling

Input: The new deployment request Ri
Output: The request queue RQ where requests sorted by the deadline

```
1: for each Rj ∈ RQ do
2:    if Ri.deadline < Rj.deadline then
3:       RQ.insert(Ri)
4:       return RQ
5:    end if
6: end for
7: RQ.append(Ri)
8: return RQ
```

Bandwidth-Aware EDF Scheduling. In addition to EDF scheduling, we design bandwidth-aware scheduling in cooperation with EDF scheduling. The key idea of bandwidth-aware scheduling is to make use of the spare bandwidth available between the local repository and the target as much as possible for parallelizing multiple requests. Thus, DDS needs the bandwidth information for each machine in the cloud. DDS would collect the bandwidth information before the whole deployment procedure begins.

Algorithm 2. Bandwidth-aware EDF scheduling

Input: *throughput* and *bandwidth* of the local repository

```
1: while throughput < bandwidth do
2:   if RQ ∉ ∅ then
3:     R_i = RQ.pop()
4:     b_j = GetBandwidth(v_i)
5:     if throughput + b_j < bandwidth then
6:       throughput = throughput + b_j
7:     else
8:       SetTransmissionRate(R_i, bandwidth − throughput)
9:       throughput = bandwidth
10:    end if
11:    StartTransmission(R_i)
12:  end if
13: end while
14: return
```

EDF is optimal when the deadlines can be satisfied. However, without bandwidth information, EDF would schedule requests in a sequential way which leads to insufficient utilization of bandwidth or even missed deployment deadlines. However if we directly schedule requests in a parallel way, the bandwidth contention among different requests can also cause deployment deadlines to be missed. Therefore, the challenge of bandwidth-aware scheduling is how to dynamically allocate transmission rates for deployment requests in order to avoid unnecessary contention. For this purpose, we design bandwidth-aware EDF algorithm as described in Algorithm 2.

As per the description of bandwidth-aware EDF, if there is spare bandwidth in the local repository, DDS will continue to obtain requests from the request queue until the required bandwidth is equal or greater than the local repository bandwidth. DDS then sets the specific rate for the last deployment request to make sure the total required bandwidth is equal to the bandwidth of local repository. Consequently, it avoids bandwidth contention with previous deployment requests and makes full use of spare bandwidth to transmit. Once a new deployment request arrives, DDS performs bandwidth-aware EDF scheduling after putting the request in the request queue. When one deployment request finishes, DDS will allocate the released bandwidth for the running requests first, and then perform bandwidth-aware EDF scheduling again.

4 Evaluation

In this section, we describe experiments for quantitative evaluation of the deadline-aware deployment system. We perform three kinds of experiments. First, we evaluate the transmission time using a DDS local repository versus a remote repository. Second, we evaluate DDS in comparison with three typical scheduling algorithms by running experiments on our cloud test-bed. Third, we evaluate DDS in larger-scale simulations.

4.1 Repository Evaluation

In this section, we compare the transmission time to a target machine from a DDS local repository and a remote repository based on Docker. In most common cases, the application provider only has the repository outside cloud. Thus, DDS would help users to create local repository within their cloud first. We provision two virtual machines with 50Mbps bandwidth in the ExoGENI Boston rack and create a local repository in one of them. Then, we use the other machine to fetch the image from the local repository and also the original remote repository (Docker Hub). The comparative results are shown in the Table 2. Note that the transmission time (T_f) from the local repository is much less than from the remote repository, the reason being that the bandwidth inside cloud is much better than outside.

Table 2. Comparison of transmission time from different repository

Docker image	Image size	Local repository	Remote repository
ubuntu	400 Mb	T_f : 8.1 s($\pm$1.1 s)	T_f : 40.8 s($\pm$2.2 s)
nginx	576 Mb	T_f : 11.7 s($\pm$1.3 s)	T_f : 58.7 s($\pm$2.5 s)
mongodb	1200 Mb	T_f : 24.4 s($\pm$1.2)	T_f : 122.4 s($\pm$3.0 s)
cassandra	1296 Mb	T_f : 26.4 s($\pm$1.5)	T_f : 132.2 s($\pm$3.1 s)

4.2 Testbed Experiments

In this section, we evaluate DDS alongside three typical scheduling algorithms in ExoGENI [2] test-bed. ExoGENI is a networked infrastructure-as-a-service (NIaaS) platform where researchers can define the network topology and bandwidth of virtual infrastructures. In our experimental setup, we chose the "xo.xlarge" type of machine as our local repository, and all other application nodes we chose "xo.medium" type machines. The guest OS in VMs which are provisioned for evaluation is Ubuntu 14.04. In the experiment, we use *iPerf* [12] to simulate the application package transmission, therefore the size of application package can be customized via *iPerf* in the evaluation. For transmission rate control, we leverage Linux Traffic Control (TC) to perform deployment request rate limiting. We use two-level Hierarchical Token Bucket (HTB) in TC: the root node classifies requests to their corresponding leaf nodes based on IP address and the leaf nodes enforce each request rate.

Schemes to Compare: We compare the following schemes with DDS.

- **FIFO:** All the deployment requests are scheduled by the arrival time of the request in a sequential way.
- **EDF:** All the deployment requests are scheduled by the EDF algorithm in a sequential way.

– **PARALLEL:** All the deployment requests are scheduled immediately after arrival in a parallel way.

Through comparison with these three schemes, we can inspect the benefits from DDS for different aspects. FIFO is the most common scheduling algorithm in distribution. EDF is optimal in sequential scheduling when the deadline can be satisfied, but it is not bandwidth-aware. PARALLEL can make high utilization of the bandwidth, but it is not deadline-aware.

Metrics: In this section, we compare the number of schedulable requests (requests that meet the deadline) and the total deployment time among different schemes. The number of schedulable requests can indicate the satisfaction of deadline requirements. The total deployment time can indicate the utilization of network bandwidth.

In this experiment, we provision two kinds of bandwidth configuration to evaluate DDS as the Table 3 described. We instantiate four nodes to deploy time critical applications in ExoGENI. For these four nodes, we generate six deployment requests which include the target machine, application size, arrival time and the deadline as the Table 4 described. To understand the scheduling mechanisms in DDS better, we assume that the installation time T_i of each application is $1s$ in this experiment.

In Fig. 3, we inspect the number of schedulable requests on different schemes. We observe that DDS can schedule more requests in two different bandwidth configurations, because sequential scheduling (EDF, FIFO) can not meet all the deadlines when multiple requests emerge simultaneously, and direct parallel scheduling suffers from bandwidth contention. Figure 4 shows the total deployment time of various schemes. We note that the total deployment time of DDS is less than EDF & FIFO, and similar to PARALLEL. This indicates that DDS makes full use of network bandwidth.

4.3 Large-Scale Simulations

Our simulations evaluate DDS considering the common public cloud provider (EC2, Azure) in this section. We evaluate the deployment schedulable ratio which is the percentage of schedulable requests in different schemes.

Table 3. Bandwidth configuration

(a) Configuration A (Mbps)				
Repository	Node1	Node2	Node3	Node4
100	20	50	70	100
(b) Configuration B (Mbps)				
Repository	Node1	Node2	Node3	Node4
100	70	70	70	70

Table 4. Deployment request

Machine	Size	Deadline	Arrival time
Node1	200 Mb	14 s	0 s
Node1	160 Mb	20 s	10 s
Node2	320 Mb	9 s	11 s
Node2	560 Mb	15 s	30 s
Node3	960 Mb	20 s	30 s
Node4	640 Mb	25 s	30 s

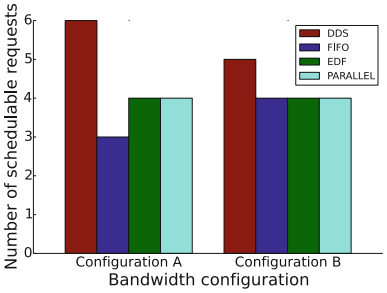

Fig. 3. Comparison of the number of schedulable requests in various schemes

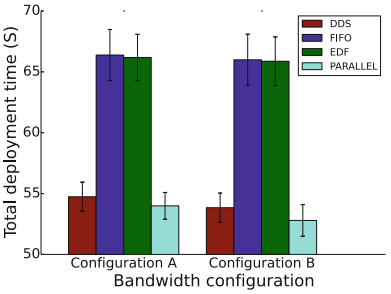

Fig. 4. Comparison of the total deployment time in various schemes

VMs Configuration: We equip the deployment server with 10 Gbps bandwidth connection and application node with 1 Gbps bandwidth connection which are typical configuration in public cloud. In the simulation, the number of application nodes range over 10, 20, 40 and 80 nodes which are sufficient to account for most distributed cloud applications.

Deployment Requests: We simulate the deployment service running 10 days ($T_{running}$) in the experiment. During this period, we generate deployment requests in different densities to simulate deploying various applications on each node. We denote S^i_{total} as the total application size of all deployment requests on node i. The request density of node i is equal to $\frac{S^i_{total}}{T_{running} * 10Gigabit}$, and the request density of whole system is the average for each node. The overall request density varies from 0.1 to 0.9. In the experiment, the deadline (d_i) of each request ranges from $10s$ to $100s$, and the application size is equal to $d_i * 1Gigabit$. We assume the installation time (T_i) is $1s$ in the simulation.

Figure 5 shows the deployment schedulable ratio in different scenarios. We observe that DDS can reduce from 24% to 83% of the deployment deadline miss ratio compared to EDF, from 26% to 89% compared to FIFO, and up to 86% compared to PARALLEL. Because EDF and FIFO schedule deployment requests in sequential way, DDS can take advantage of parallelized deployments. The PARALLEL scheme parallelizes deployments but suffers severe bandwidth contention as request density increases. In contrast, DDS is bandwidth-aware and provides dynamic transmission rate control to avoid bandwidth contention for different deployment requests. In summary, DDS significantly reduces the number of deadline missing requests for deploying cloud applications.

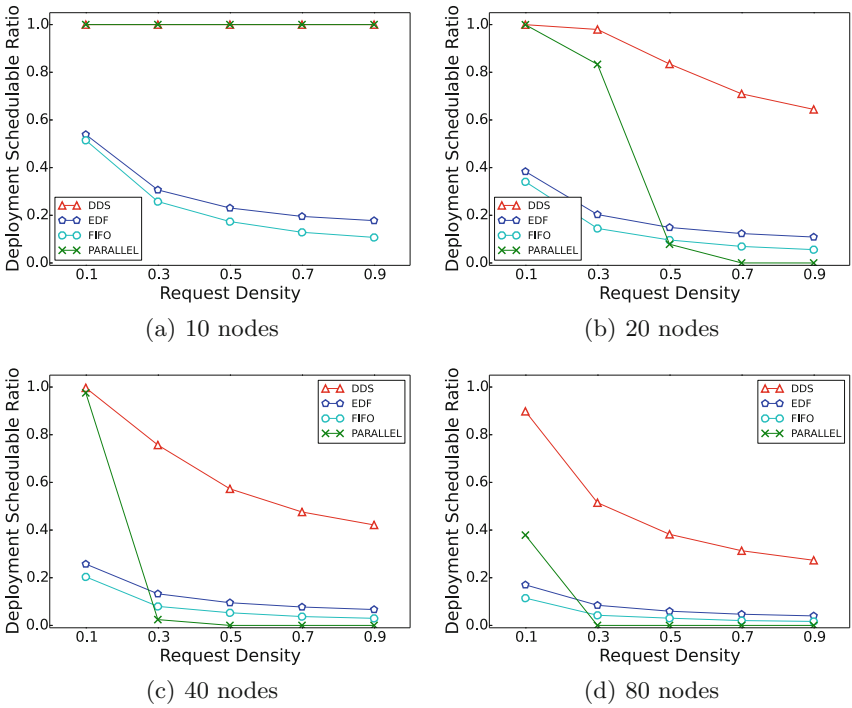

Fig. 5. Comparison of the deployment schedulable ratio in different scenarios

5 Related Work

In recent years, deployment has been an important topic in distributed environment, service-oriented systems and cloud computing. The techniques in DDS are related to the following areas of research:

Automatic Cloud Application Deployment. To enable automatic deployment has been the focus of several recent works. SO-MVDS [5] allows users to design and create virtual machines with specific services running in them and define a service deployment request to enhance the efficiency of service deployment. Li et al. [7] propose a general approach to application deployment. They adopt contextualization process which is to embed various scripts in VM images to initiate applications. DDS, on the other hand, is compatible with Docker containers, achieving automatic deployment more easily.

On-Demand Image Distribution. The idea of distributing images in clouds efficiently has been explored in recent works. Vaquero et al. [15] proposes a solution based on combining hierarchical and Peer to Peer (P2P) data distribution techniques. VDN [10], a new VM image distribution network on the top of chunk-level, enables collaborate sharing in cloud data centers. These approaches focus

on fast transmission. In contrast, DDS is not only transmitting images efficiently but is also aware of deadlines via scheduling mechanisms.

Deadline-Aware Scheduling Techniques. D^3[17] and D^2TCP [14] are transport protocols designed for deadline-aware transmission inside data centers. These protocols add the deadline information to TCP and provide control mechanisms based on the deadline information. Techniques like Karuna [4] and pFabric [1] prioritize network flows to transmit. All these approaches schedule transmission at flow level. In contrast, DDS exploits the information of bandwidth to schedule transmission in application level which is more relevant to users requirements.

6 Conclusion

It is challenging to deploy time critical applications into clouds while meeting the time constraints of deployment. This is an important and practical problem, but has been neglected by prior work in this field. In this paper, we propose a Deadline-aware Deployment System (DDS) which helps users to create local repository and automatically deploy applications into clouds. We investigate the scheduling mechanisms in cloud deployment system and implement bandwidth-aware EDF scheduling algorithm in DDS. DDS schedules deployment requests based on deadline and bandwidth information to make better scheduling decision. In the evaluation, we showed that DDS leverages network resources sufficiently and significantly reduces the number of missed deployment deadlines. Furthermore, we plan to investigate multiple repositories deployment and inter-data center network for time critical cloud applications.

Acknowledgments. This research has received funding from the European Union's Horizon 2020 research and innovation program under grant agreements 643963 (SWITCH project), 654182 (ENVRIPLUS project) and 676247 (VRE4EIC project). The research is also partially funded by the COMMIT project.

References

1. Alizadeh, M., Yang, S., Sharif, M., Katti, S., McKeown, N., Prabhakar, B., Shenker, S.: pFabric: minimal near-optimal datacenter transport. In: ACM SIGCOMM Computer Communication Review, vol. 43, pp. 435–446. ACM (2013)
2. Baldin, I., Chase, J., Xin, Y., Mandal, A., Ruth, P., Castillo, C., Orlikowski, V., Heermann, C., Mills, J.: ExoGENI: a multi-domain infrastructure-as-a-service testbed. In: McGeer, R., Berman, M., Elliott, C., Ricci, R. (eds.) The GENI Book, pp. 279–315. Springer, Cham (2016). doi:10.1007/978-3-319-33769-2_13
3. Casalicchio, E., Silvestri, L.: Mechanisms for SLA provisioning in cloud-based service providers. Comput. Netw. **57**(3), 795–810 (2013)
4. Chen, L., Chen, K., Bai, W., Alizadeh, M.: Scheduling mix-flows in commodity datacenters with Karuna. In: Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference, pp. 174–187. ACM (2016)

5. Gao, W., Jin, H., Wu, S., Shi, X., Yuan, J.: Effectively deploying services on virtualization infrastructure. Front. Comput. Sci. **6**(4), 398–408 (2012)
6. Hu, Y., Li, H., Peng, Y.: NVLAN: a novel VLAN technology for scalable multitenant datacenter networks. In: 2014 Second International Conference on Advanced Cloud and Big Data (CBD), pp. 190–195. IEEE (2014)
7. Li, W., Svärd, P., Tordsson, J., Elmroth, E.: A general approach to service deployment in cloud environments. In: 2012 Second International Conference on Cloud and Green Computing (CGC), pp. 17–24. IEEE (2012)
8. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM (JACM) **20**(1), 46–61 (1973)
9. Merkel, D.: Docker: lightweight Linux containers for consistent development and deployment. Linux J. **2014**(239), 2 (2014)
10. Peng, C., Kim, M., Zhang, Z., Lei, H.: VDN: virtual machine image distribution network for cloud data centers. In: 2012 Proceedings IEEE INFOCOM, pp. 181–189. IEEE (2012)
11. Smith, W., Foster, I., Taylor, V.: Predicting application run times using historical information. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1998. LNCS, vol. 1459, pp. 122–142. Springer, Heidelberg (1998). doi:10.1007/BFb0053984
12. Tirumala, A., Qin, F., Dugan, J., Ferguson, J., Gibbs, K.: Iperf: the TCP/UDP bandwidth measurement tool (2005). http://dast.nlanr.net/Projects
13. Tsai, W., Bai, X., Huang, Y.: Software-as-a-service (SaaS): perspectives and challenges. Sci. China Inf. Sci. **57**(5), 1–15 (2014)
14. Vamanan, B., Hasan, J., Vijaykumar, T.: Deadline-aware datacenter TCP (D2TCP). ACM SIGCOMM Comput. Commun. Rev. **42**(4), 115–126 (2012)
15. Vaquero, L.M., Celorio, A., Cuadrado, F., Cuevas, R.: Deploying large-scale datasets on-demand in the cloud: treats and tricks on data distribution. IEEE Trans. Cloud Comput. **3**(2), 132–144 (2015)
16. Wang, J., Taal, A., Martin, P., Hu, Y., Zhou, H., Pang, J., de Laat, C., Zhao, Z.: Planning virtual infrastructures for time critical applications with multiple deadline constraints. Future Gener. Comput. Syst. (2017)
17. Wilson, C., Ballani, H., Karagiannis, T., Rowtron, A.: Better never than late: meeting deadlines in datacenter networks. In: ACM SIGCOMM Computer Communication Review, vol. 41, pp. 50–61. ACM (2011)
18. Zhao, Z., Martin, P., De Laat, C., Jeffery, K., Jones, A., Taylor, I., Hardisty, A., Atkinson, M., Zuiderwijk, A., Yin, Y., Chen, Y.: Time critical requirements and technical considerations for advanced support environments for data-intensive research. In: 2nd International Workshop on Interoperable Infrastructures for Interdisciplinary Big Data Sciences (IT4RIs) in the Context of IEEE Real-Time System Symposium (RTSS) (2016)
19. Zhao, Z., Martin, P., Wang, J., Taal, A., Jones, A., Taylor, I., Stankovski, V., Vega, I.G., Suciu, G., Ulisses, A., et al.: Developing and operating time critical applications in clouds: the state of the art and the SWITCH approach. Procedia Comput. Sci. **68**, 17–28 (2015)
20. Zhou, H., Hu, Y., Wang, J., Martin, P., De Laat, C., Zhao, Z.: Fast and dynamic resource provisioning for quality critical cloud applications. In: 2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC), pp. 92–99. IEEE (2016)