Online Work Distribution to Clouds

Lan Wang, Olivier Brun, Erol Gelenbe

Online Work Distribution to Clouds

Lan Wang
Imperial College London
Email: lan.wang12@imperial.ac.uk

Olivier Brun
LAAS/CNRS
Email: brun@laas.fr

Erol Gelenbe, *FIEEE*
Imperial College London
Email: e.gelenbe@imperial.ac.uk

***Abstract*—Cloud systems include both locally based servers at user premises and remote servers and multiple Clouds that can be reached over the Internet. This paper describes a smart distributed system that combines local and remote Cloud facilities. It operates with a task allocation system that takes decisions to allocate tasks dynamically to the service that offers the best overall Quality of Service and a routing overlay which optimizes network delay for data transfer between clouds. Experimental results are conducted at the global intercontinental level, both to collect data for decision making and to illustrate the effectiveness of our approach.**

***Index Terms*—Cloud Computing, Adaptive Networked Systems, Quality of Service, Performance Management, Global Internet, World-Wide Experiments, Smart Intercontinental Networks**

I. Introduction

Cloud computing enables the consolidation of an increasing number of applications from the general public or enterprise users which generate diverse sets of workloads in terms of resource demands and performance requirements [1]. For example:

- Web requests usually demand fast response and produce loads that may vary significantly over time [2];
- Scientific applications are commonly computation intensive and might undergo several phases with varied workload profiles [3];
- MapReduce jobs consist of different tasks of various sizes and resource requirements [2], [4].

Moreover, the heterogeneity in the hardware configuration of physical servers or virtual machines in terms of the specific speeds and capacities of the processor, memory, storage, and networking subsystems further complicates the matching of applications to available machines.

Therefore, it is really challenging for Cloud service providers to dispatch incoming tasks to servers with the assurance of the quality and reliability of the job execution required by end users while also improving efficiency in the usage of resources.

Furthermore, the wide-spread usage of IP networks allows large-scale Cloud providers such as Amazon EC2[5], Microsoft Windows Azure[6] and Google Compute Engine[7] to build Cloud infrastructures at a global scale which facilitates the deployment of applications spanning multiple regions around the world for improving reliability and provisioning services with lower latency and a better experience for the end users. This enables service providers to distribute workloads across multiple Clouds for balancing load and offering better Quality of Service(QoS) to all the requests it receives. The selection of a Cloud for an end user also relies on other considerations such as security, cost, and energy consumption. For example, cloud bursting is "an application deployment model in which an application runs in a private cloud (on local servers) and bursts into a public cloud when the demand for capacity spikes"[8].

Using the Internet as the network medium between Cloud regions is the default choice offered by many Cloud providers. It is known that the routes set up by IP do not necessarily result in the best performance [9], and that IP connections may have lower reliability than other paths [10], [11], [12], [13]. The Cloud processing latency is affected by workload distribution inside the Cloud; this not always easy to assess due to the heterogeneity in both workload and machine hardware, and the dynamic changes of load conditions over time. Much research on the solutions to these issues require sophisticated optimisation algorithms with relatively high computational complexity [14], [15], [16], [17]. Some approaches are also based on substantial performance measurements resulting in traffic overhead [18], [19], [20], [21], [22] and potential overhead in the Cloud. Such approaches can be difficult to put into practice due to scalability problems for large systems and the need for computational efficiency when one must make real-time and on-line decisions.

A. Approach of the current paper

In recent work [23] we have examined how a task allocation platform (TAP) that is *internal* to a Cloud can be used to dynamically make task allocation decisions to optimise QoS, and have suggested both model based and learning based approaches. In other recent work [24] we have shown that a limited amount of re-routing over an overlay network using a measurement and big data approach can substantially average end-to-end delay for internet traffic and also reduce the perceived packet loss.

Here we combine these two ideas. In this paper we focus on the QoS that tasks receive, in particular the overall response time which is determined by the network latency to access, forward data and programs, retrieve results and data, and which includes the local or remote Cloud processing delay. In particular, we propose a practical system for adaptive workload distribution across multiple Clouds over wide area networks. The system includes a TAP deployed in each Cloud that optimises user perceived QoS and exploits the routing

overlay $SMART$ over Clouds [24] for improving network delay incurred by data transfer.

In this approach, user's requests are routed to a designated local Cloud, which may well be the geographically closest one, provided that it has enough available capacity to handle the request. When the workload at the local Cloud increases, the TAP at the local Cloud can decide to forward requests to remote Clouds. In the process, TAP will consider the effect of both the data transfer delay and Cloud processing delay, each being weighted for their relative importance. The estimate of data transfer delay used in our system also takes into account the measured packet loss which will result in extra network delay for applications that use TCP for data transmission.

In order to optimise Cloud delay and network latency, our approach is meant to be easily deployable over a large population of machines, and it should be able to make fast online decisions resulting in good quality of service (QoS) with low computational overhead. Although it requires measurement and monitoring both of network characteristics and of local and remote Cloud delays, we limit the frequency and overhead related to the monitoring effort, and also limit the computational complexity of decision making using reinforcement learning [25].

We therefore propose an approach, applied both in TAP and to the routing overlay, which uses Reinforcement Learning [25] with the random neural network [26], [27], [28], [29] to make fast, judicious and efficient decisions based on the knowledge learned from the past observations, while adapting to changes in workload and on-going performance of the Cloud environment. Our approach benefits from limited measurement overhead as it probes the performance of subsystems which provide better QoS, while still exploring less frequently a wider range of alternative systems that can in the future prove to provide improved QoS if the current set of frequently used subsystems result in poor QoS.

Experiments were conducted on a real large scale system operating on the Internet at a global scale and we empirically evaluate the potential of our proposed algorithms when there is great diversity both in the types of jobs, the class of QoS criteria and the resources they request. The experimental results that we obtain, validate the adaptiveness and effectiveness of our proposed system for dynamic environments.

II. Learning using Random Neural Network

In this section, we present the algorithm designed for making optimized routing decisions used both in TAP and the routing overlay. A Random Neural Network (RNN) comprises N fully connected neurons [29], where each neuron i $(i = 1, 2, ...N)$ is characterised by an integer $k_i(\tau) \geq 0$ which is its "level of excitation", τ represents time. A neuron fires at time τ, if $k_i(\tau) > 0$. Each neuron can receive positive and negative signals (spikes) either from other neurons or from the outside world, which increase or decrease the k_i of the receiving neuron. It has been proved [27], [30] that, at the equilibrium state, the probabilities:

$$q_i = \lim_{\tau \to \infty} Prob[k_i(\tau) > 0], \tag{1}$$

are uniquely obtained from the expression:

$$q_i = \frac{\Lambda(i) + \sum_{j=1}^{N} q_j w^+(j,i)}{r(i) + \lambda(i) + \sum_{j=1}^{N} q_j w^-(j,i)}, \tag{2}$$

where the $w^+(j,i)$ and $w^-(j,i)$ are the excitatory and inhibitory weights from neuron j to neuron i ($w^+(i,i) = w^-(i,i) = 0$), and $\Lambda(i)$ and $\lambda(i)$ are the inputs of external excitatory and inhibitory signals to neuron i, while the firing rate at a neuron:

$$r(i) = \sum_{j=1}^{N} [w^+(i,j) + w^-(i,j)] \tag{3}$$

In TAP, a distinct RNN corresponds to a job class which has a distinct goal function G defined based on user desired QoS. Each neuron of a RNN corresponds to the choice of a machine in a cluster for accommodating jobs.

A given RNN is initialised by setting all the inter-neuron weights $w^+(i,j) = w^-(i,j) = 1/2(N-1)$, so that $r(i) = 1$ for all i, and $\Lambda(i) = 0.25 + 0.5\lambda(i)$. In particular we can choose $\lambda(i) = 0$ so that all $\Lambda(i) = 0.25$. This of course results in $q_i = 0.5$ for all i. Then, the value of q_i is changed using (2) with the successive updates of the weights based on the measured value of the goal function.

Suppose TAP receives a value G_i^t of the goal function that was measured at time t at host i, the inverse of the goal function is defined as the reward $R_i^t = \frac{1}{G_i^t}$. The RNN weights are updated as follows:

- We first update a decision threshold T as

$$T \leftarrow \alpha T + (1-\alpha) R_i^t \tag{4}$$

  where $0 < \alpha < 1$ is a parameter used to vary the relative importance between "past history" and the most recent measurement.
- Then, if $R_i^t > T$, it is considered that the previous decision made by the RNN was successful and the weights are updated as follows:

$$\begin{aligned} w^+(j,i) &\leftarrow w^+(j,i) + R_i^t \\ w^-(j,k) &\leftarrow w^-(j,k) + R_i^t/(N-2),\ if\ k \neq i \end{aligned}$$

- $else\ if\ R_i^t < T$

$$\begin{aligned} w^+(j,k) &\leftarrow w^+(j,k) + R_i^t/(N-2),\ if\ k \neq i \\ w^-(j,i) &\leftarrow w^-(i,j) + R_i^t, \end{aligned}$$

After the weights are updated, the q_i are computed using (2) iteratively until it is converged. The algorithm tends to increase the value of q_i to neuron i, $i = 1, \ldots, N$ which corresponds to a host that has a smaller measurement value of G, so that each time TAP selects the host corresponding to the largest q_i, resulting in the minimizing of the QoS goal.

In $SMART$ (the routing overlay we designed) [24], the algorithm uses the RNN to choose a subset of paths to probe

at each successive time slot, and measures the sum of edge delays in the probed paths. The algorithm then selects the minimum latency path among those it has probed.

III. Task Allocation Platform for a Single Cloud

In our early work[23], We developed a practical Task Allocation Platform (TAP), which is a Linux based portable software module and can be easily installed on a machine with Linux OS, to accommodate the distinct static or dynamic allocation algorithms and perform online measurement. To use TAP, users only need to declare QoS goals such as fastest job execution or optimising cloud provider's profit while maintaining service level agreements (SLAs). TAP accepts these directions and carries out constant monitoring and measurement in order to keep awareness of the state of cloud environment and service performance related to the QoS goals. With the knowledge learned from these observations, the task allocation algorithms hosted in the TAP make online decisions to achieve the best possible QoS requested by users, while adapting to conditions that vary over time.

The TAP (as shown in Figure 3) has a centralized controller accommodating the online task allocation algorithms and running on a dedicated host. It penetrates into the cloud infrastructure by embedding measurement agents into each machine to conduct self-observation that are relevant to the required QoS.

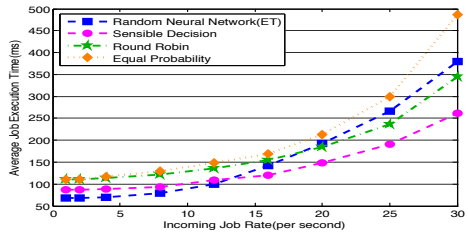
We conducted experiments to evaluate the adaptiveness and effectiveness of our proposed approach using the RNN-based algorithm described in section II and the sensible algorithm which is also an on-line and adaptive strategy which direct a incoming job to a host with the probability which is inversely proportional to the value of a goal function updated by the most recent measurement.

Experiments were conducted on a multiple host test-bed including three hosts for processing jobs. The choice of the test-bed in the small scale is in that we can easily vary the load of the system by varying arrival rates of tasks.

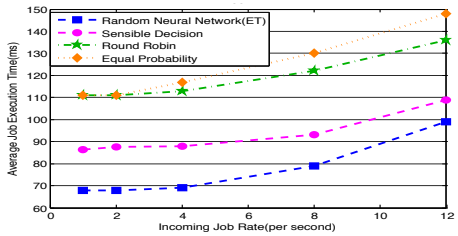
Actually, TAP is scalable because it only makes a limited measurement effort by focusing on the monitoring of better performed servers, and it explores the other machines with lower probability (e.g. 10%). The job used in the first set of experiments is a "prime number generator" which is deployed in advance on the host, so that it actually provides the prime number generating services which generate CPU intensive workload. The QoS goal initially considered was the minimisation of the execution time on the host.

To build a heterogeneous server environment, we introduce a *background load* on each host which stresses the CPU distinctly on the three host $i = 1,\ 2,\ 3$ with relative processing speeds of $2\ :\ \ 4\ :\ \ 1$ for the CPU intensive services we provided. The experiment was repeated for a range of average task arrival rates λ, in order to evaluate performance under load conditions that vary from light to heavy load, including saturation.

The results in Figure 1 show that our proposed algorithm, the RNN and the sensible algorithm, benefits from their potential to detect the performance differences between the servers by leaning from the measurements, and direct tasks to the hosts which provide a better performance, outperforming Round Robin and equal probability task allocation which are commonly-practiced schemes. The RNN-based algorithm clearly outperforms the others (in Figure 1(b)), confirming that it is a fine-grained QoS-aware online task allocation algorithm.



(a)



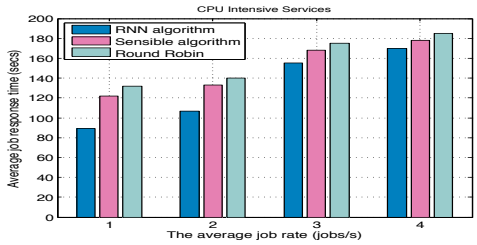
(b)

Fig. 1. Average execution time experienced in a cluster composed of hosts with distinct processing capacities.

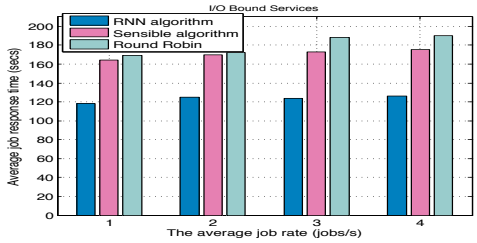
To investigate the potential of our proposed algorithms when there is greater diversity both in the types of jobs, the class of QoS criteria and the resources they request, we introduce a web browsing workload generated using HTTPerf which is a web server performance tool. It originates HTTP requests which retrieve files from web server, such as Apache 2 HTTP server, thereby generating I/O bound workload on the web server without much CPU consumption.

We introduce a background load which stresses I/O distinctly on each host, resulting in relative processing speeds of $6\ :\ 2\ :\ 1$ with respect to I/O bound services, while a background load which stresses CPU differently, resulting in the relative processing speed of $2\ :\ 3\ :\ 6$ for the CPU bound services.

The results in Figure 2 show that the RNN based algorithm performs particularly better due to its potential to make more accurate decisions (compared with Sensible) which direct I/O bound tasks to the hosts which provide better I/O capacity and transfer CPU intensive tasks to the hosts with higher processing speed. In the experiments, we also reduced the background load in terms of both CPU and I/O stress on the $Host\ 2$ to the lowest level as compared with the $Hosts\ 1,\ 3$. The RNN based algorithm was found to be able to detect the load changes and dispatch the majority of subsequent tasks of both classes to $Host2$, which also shows the host



(a)



(b)

Fig. 2. Average response time experienced by CPU intensive services and I/O bound services in a heterogeneous cluster. We compare Round-Robin with RNN based Reinforcement Learning and the Sensible Algorithm.

which is heavily loaded in terms of CPU can still offer good performance to I/O bound tasks, thereby improving the resource utilization.

IV. Task Allocation Platform for Multiple Clouds

In this section, we present an extension of the TAP designed for workload distribution across multiple clouds over wide area networks. As shown in Figure 3, users web requests are routed to the local, which is perhaps the geographically closest cloud, if the cloud has enough capacity. The dispatcher at the local cloud receives the incoming workload and adaptively selects the best possible server based on the user-defined QoS (e.g. the response time).

If the workload in the local cloud increases, the dispatcher could decide whether to forward the subsequent requests to the remote clouds in order to balance the load and offer better QoS to all the requests it receives. The decision would rely on a variety of considerations, such as security, cost, QoS and energy consumption.

We currently only concern ourselves with the QoS that tasks receive, in particular the response time observed by tasks. Obviously this response time will be determined by the network delay incurred by the access to local or remote clouds, which includes the network delay to process the request, the network delay to forward the task with its possible code and data, plus the time it takes to return the results to the sender after execution, plus the waiting time and service time inside the clouds. That is to say, the dispatcher would select a remote cloud (say the j-th cloud) to share the workload of the local cloud by considering the response time (denoted by D^j_{job}) which consists of the data transfer latency (denoted by D^j_n)

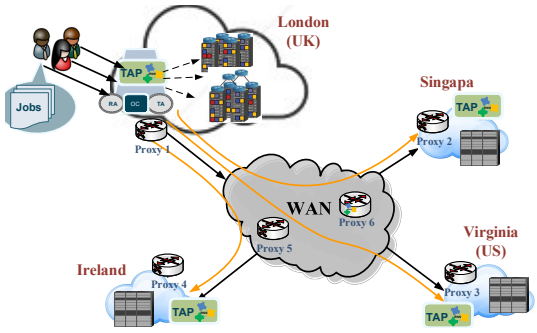


Fig. 3. The Architecture of the TAP for workload distribution across multiple clouds over wide area networks

which depends on the traffic conditions on the connections to the remote clouds and the proximity of the remote cloud to the local cloud as well as the response time within the cloud (denoted by D^j_e) which is related to the viability or health state of the cloud. Hence, a goal function for the decision can be presented as,

$$D^j_{job} = aD^j_n + bD^j_e \tag{5}$$

where a and b are the relative importance being placed on the data transfer latency on network connections and the processing delay inside the cloud. It should be noted that the data transfer latency for the local cloud, though not zero, may be negligible. We assume that each cloud will report its health state regularly, including the response time within the cloud. In the following section, we present the approaches we used to predict the data transfer latency via measuring the network delay on the connections to all the external clouds. This of course is easier said than done because it depends on the nature of the transfers and on the other traffic in the connections.

A. Data transfer delay estimation

To simplify matters, we measure the round-trip delay (denoted by $T^j_p(t)$ for the j-th cloud at time t) and the packet loss (denoted by $L^j(t)$) on each connection via pinging every a certain time interval (e.g. 1 second) thereby obtaining the network delay. A weighted average of the measurements for the round-trip delay is used as

$$T^j_p = \alpha T^j_p + (1-\alpha)T^j_p(t) \tag{6}$$

where $0 < \alpha < 1$ is a parameter which reflects the relative importance between the past values and the most recent measurement.

The packet loss needs to be considered because some applications (e.g. HTTP requests) utilize TCP to transfer the data and require retransmissions for the lost packets, which results in extra delay. The loss is measured as follows:

$$L^j(t+1) = \frac{\alpha t L^j(t) + (1-\alpha)t1_{[loss\ on\ the\ t-th\ ping]}}{t+1} \tag{7}$$

It should be noted that there is a loss on the connection to the j-th cloud if the pinging delay at the j-th cloud T_p^j is larger than some fixed value $T_{timeout}$.

Therefore, the network delay on the connection to the j-th cloud considering packet loss can be expressed as

$$D_p^j = L^j(D_p^j + T_p^j) + (1 - L^j)T_p^j \tag{8}$$

where D_p^j appears on both sides of eq. (8) because the retransmission of a lost packet might on the average surfer the same network delay D_p^j .

Therefore, we finally get

$$D_p^j = \frac{T_p^j}{1 - L^j} \tag{9}$$

The data transfer time of a request to a remote cloud and its response traveling back to the local cloud can be approximated using the corresponding network delay which is obtained from the ongoing measurements on the same connection:

$$D_n^j = (\frac{S}{M})D_p^j = \frac{ST_p^j}{M(1 - L^j)} \tag{10}$$

where M is the maximum packet size (bytes), and S is the total data size (bytes).

Upon the arrival of each job at TAP, the allocation decision is made using the simple greedy algorithm which selects the cloud offering the minimal response time.

$$\arg\min_j (a\frac{ST_p^j}{M(1 - L^j)} + bD_e^j) \tag{11}$$

B. Routing Overlay for Improving Data Transfer Performance

To improve data transfer performance on the connections between a local cloud and remote clouds, we proposed a routing overlay, "SMART"[24], which is a self-healing, self-optimizing and highly scalable routing overlay that accepts customized routing policies formulated by distributed applications according to their own needs. It can be widely deployed over a sizable population of routers.

The overlay we designed is based on a set of proxies (Figure 3) installed at different Cloud servers, or they may be in other servers across the network, it operates by monitoring the quality of Internet paths (latency, bandwidth, loss rate) between overlay nodes and rerouting packets along an alternate path when the primary becomes unavailable or suffers from congestion, while the flow of packets between proxies travels in conventional IP mode.

The routing decisions are made on-line at each proxy of the overlay network based on adaptive learning techniques using the random neural network (RNN). By using this algorithm, probing does not cover all possible paths but only a few paths which have been observed in previous probing steps to provide low overall forwarding delay for packets, so that it uses a limited monitoring effort but achieves asymptotically the same average (per round) end-to-end latency as the best path. Therefore, it can be widely deployed over a sizable population of routers.

C. Experimental results

To validate our proposed system, we built an experimental system at the global intercontinental level which includes the local cloud in the test-bed of Imperial College London and the three remote clouds located in Ireland, Virginia and Singapore provided by Amazon AWS.

The web requests are originated using Httperf for retrieving a file of size 128K from the web servers, which generates I/O bound workload on the web servers. In the local cloud, the TAP is deployed for optimizing the web request allocation across the three web servers with distinct I/O capacity. In the remote clouds, there are other web servers deployed for load balancing. Between the local and remote clouds, the routing overlay–SMART–is used for routing the web requests and responses with the optimized network delay.

The measurement of the network delay on the connections to all the remote clouds is carried out every one second and the average response time for the Httperf requests inside each cloud is also reported every one second. As shown in Figure 4, the network connection to the cloud located in Ireland has the lowest network delay because this cloud is closer to the local cloud in London than the others, and the traffic on the connection appears to be low.

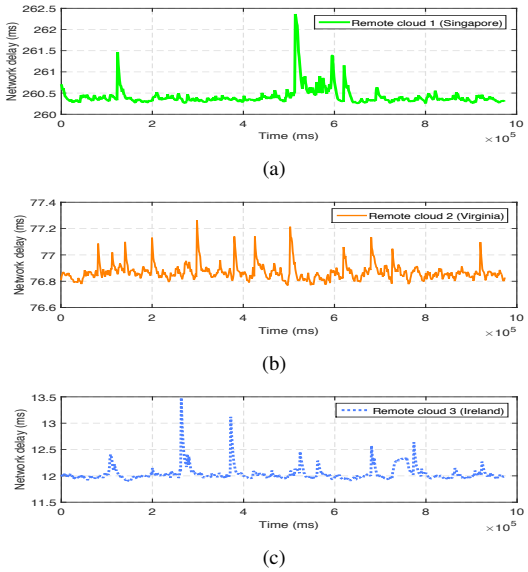


Fig. 4. The weighted average network delay over time on the connections to the three remote clouds; it is derived from the measurement of the round-trip delay and loss via pinging.

The web browser requests were originated from a client inside the local cloud at the rate of 0.5 requests per second. We varied the background workloads, which consume I/O capacity on the web servers in the local cloud over time in order to observe whether our TAP is able to adapt to the changing load conditions. Therefore, the local cloud is loaded lightly, modestly, heavily and finally lightly during 200 seconds for each of these successive conditions.

As shown in Figure 5, the incoming web requests were

dispatched to the local cloud when it was under light and modest load conditions. Our autonomic TAP learned the optimal request allocation based on the online measurement and directed the requests to the web server which provided the fastest response. As the workload in the local cloud increased to a certain high level which resulted in a response time that was significantly greater than that of one of the remote clouds (including network delays), the TAP selected the remote cloud for the subsequent web requests until it detected that the local clouds time had dropped significiantly due to the offloading of workload from background tasks.

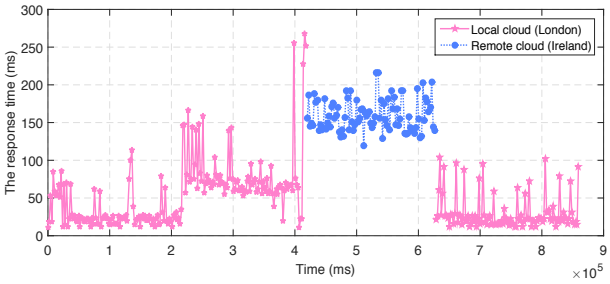


Fig. 5. The measured response time from our experiment, measured for each web request as time elapses; the different colours represent the different clouds where the requests are allocated and processed.

The improvement of network performance achieved using our overlay network is shown in Figures 6 and 7. We deployed our overlay network across Europe, Asia, North and South America, and Australia using 19 overlay nodes. The RTT observed between Japan and Chile using the direct IP route was roughly 400 ms, whereas the RTT of the minimum latency path obtained with our overlay based adaptive routing was approximately 250 ms.

As can be seen, the RNN-based overlay routing algorithm learns very quickly which path has the minimum latency path, and tracks it throughout the 5-day experiment. Similarly, the available throughput between Sydney (Australia) and Virginia (USA) using the direct IP route was 8.5 Mbps, whereas the average throughput of the optimal path turned out to be 55.3 Mbps.

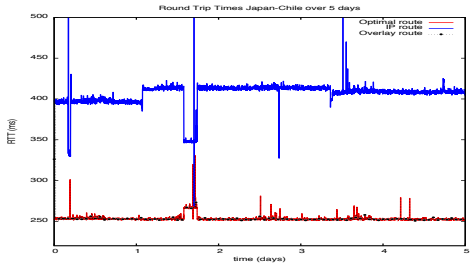


Fig. 6. RTD in milliseconds measured for the Japan-Chile connection in an experiment lasting 5 successive days.

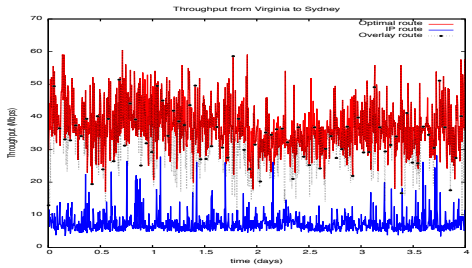


Fig. 7. Throughput (Mbps) measured from Virginia (USA) to Sydney (Australia) over 4 consecutive days.

V. Conclusions

This paper proposes a practical system for adaptive workload distribution across multiple clouds over wide area networks. It uses a Random Neural Network (RNN) based adaptive learning algorithm with Reinforcement Learning to optimize the cloud processing and network delay to adaptively distribute workload based on both cloud delay and data transfer latency which is estimated using network delay.

Experiments conducted on a real large scale system operating across the Internet at a global scale validate the adaptiveness and effectiveness of our proposed system in dynamic environments. In future work, we will also take into account the energy consumption of jobs to take descisons for load distribution among local and remote Clouds.

Acknowledgments

We gratefully acknowledge the financial support of the EC 7th Framework Programme's PANACEA Project (www.panacea-cloud.eu) under Grant Agreement No. 610764.

References

[1] C. Delimitrou and C. Kozyrakis, "Qos-aware scheduling in heterogeneous datacenters with paragon," *ACM Trans. Comput. Syst.*, vol. 31, no. 4, pp. 12:1–12:34, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2556583

[2] J. Zhan, L. Wang, X. Li, W. Shi, C. Weng, W. Zhang, and X. Zang, "Cost-aware cooperative resource provisioning for heterogeneous workloads in data centers," *Computers, IEEE Transactions on*, vol. 62, no. 11, pp. 2155–2168, Nov 2013.

[3] A. Iosup, S. Ostermann, M. Yigitbasi, R. Prodan, T. Fahringer, and D. H. J. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 6, pp. 931–945, June 2011.

[4] B. Palanisamy, A. Singh, and L. Liu, "Cost-effective resource provisioning for mapreduce in a cloud," *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[5] "Amazon ec2," http://aws.amazon.com/ec2/.

[6] "Windows azure," http://www.windowsazure.com/.

[7] "Google compute engine," https://cloud.google.com/compute/.

[8] "cloud bursting," http://searchcloudcomputing.techtarget.com/definition/cloud-bursting.

[9] V. Paxson, "End-to-end routing behavior in the internet," in *in Proc. ACM SIGCOMM'96*, Stanford, CA, USA, August 1996, pp. 25–38.

[10] C. Labovitz, R. Malan, and F. Jahanian, "Internet routing instability," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 515–526, 1998.

[11] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed internet routing convergence," *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 175–187, Aug. 2000. [Online]. Available: http://doi.acm.org/10.1145/347057.347428

[12] M. Dahlin, B. Chandra, L. Gao, and A. Nayate, "End-to-end wan service availability," in *In Proc. 3rd USITS*, 2001, pp. 97–108.

[13] J. Han and F. Jahanian, "Impact of path diversity on multi-homed and overlay networks," in *In Proceedings of IEEE International Conference on Dependable Systems and Networks*, 2004.

[14] H. Topcuouglu, S. Hariri, and M. you Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002. [Online]. Available: http://dx.doi.org/10.1109/71.993206

[15] S. Pandey, W. Linlin, S. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, April 2010, pp. 400–407.

[16] S. Zaman and D. Grosu, "A combinatorial auction-based dynamic vm provisioning and allocation in clouds," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, Nov 2011, pp. 107–114.

[17] E. Hou, N. Ansari, and H. Ren, "A genetic algorithm for multiprocessor scheduling," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 5, no. 2, pp. 113–120, Feb 1994.

[18] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '01. New York, NY, USA: ACM, 2001, pp. 131–145. [Online]. Available: http://doi.acm.org/10.1145/502034.502048

[19] L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," in *in Proceedings of the 3rd ACM Workshop on Hot Topics in Networks (HotNets-III)*, November 2004.

[20] J. Touch, Y. Wang, L. Eggert, and G. Finn, "A virtual internet architecture," ISI, Tech. Rep. ISI-TR-2003-570, March 2003.

[21] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe, "The case for separating routing from routers," in *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, A. Press, Ed., 2004.

[22] M. Beck, T. Moore, and J. Plank, "An end-to-end approach to globally scalable programmable networking," in *in Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, A. Press, Ed., 2003.

[23] L. Wang and E. Gelenbe, "Adaptive dispatching of tasks in the cloud," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2015.

[24] O. Brun, L. Wang, and E. Gelenbe, "Big data for autonomic intercontinental overlays," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. 575–583, March 2016.

[25] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[26] E. Gelenbe and S. Timotheou, "Random neural networks with synchronized interactions," *Neural Computation*, vol. 20, no. 9, pp. 2308–2324, 2008. [Online]. Available: http://dx.doi.org/10.1162/neco.2008.04-07-509

[27] E. Gelenbe, "Stability of the random neural network model," *Neural Computation*, vol. 2, no. 2, pp. 239–247, 1990.

[28] ——, "Random neural networks with negative and positive signals and product form solution," *Neural Computation*, vol. 1, no. 4, pp. 502–510, Dec 1989.

[29] E. Gelenbe and J. Fourneau, "Random neural networks with multiple classes of signals," *Neural Computation*, vol. 11, no. 4, pp. 953–963, 1999.

[30] E. Gelenbe and S. Timotheou, "Synchronized interactions in spiked neuronal networks," *The Computer Journal*, vol. 51, no. 6, pp. 723–730, 2008.