Adaptive Allocation of Multi-class Tasks in the Cloud

Lan Wang

Imperial College London, UK
lan.wang12@imperial.ac.uk

Abstract. Cloud computing enables the accommodation of an increasing number of applications in shared infrastructures. The routing for the incoming jobs in the cloud has become a real challenge due to the heterogeneity in both workload and machine hardware and the changes of load conditions over time. The present paper design and investigate the adaptive dynamic allocation algorithms that take decisions based on on-line and up-to-date measurements, and make fast online decisions to achieve both desirable QoS levels and high resource utilization. The Task allocation platform(TAP) is implemented as a practical system to accommodate the allocation algorithms and perform online measurement. The paper studies the potential of our proposed algorithms to deal with multi-class tasks in heterogeneous cloud environments and the experimental evaluations are also presented.

Keywords: Random Neural Network · Reinforcement Learning · Sensible Algorithm · Task allocation · Cloud Computing · task dispatching

1 Introduction

Cloud computing enables the inhabitation 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 [2]. For example, Web requests usually demand fast response and produce loads that may vary significantly over time [18]; Scientific applications are commonly computation intensive and might undergo several phases with varied workload profiles [10]; MapReduce jobs consist of different tasks of various sizes and resource requirements [18]. The consolidation of highly heterogeneous workloads in shared IT infrastructure causing inevitable interference between co-located workloads [20] can also degrade performance. Furthermore, 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 improving resource usage efficiency.

Much research have been carried out on task allocation approaches for QoS improvement in the cloud, such as DAC-based modeling [14, 11], genetic algorithms [9], colony optimization (ACO) [1], Particle Swarm Optimization [13], Random Neural Networks [7], and auction-based mechanisms [17]. The authors in [2, 12] focus on modeling the diversity among applications on heterogenous servers. The trade-off between energy consumption and QoS was addressed in [5, 6, 19, 15]. Workload distribution across multiple clouds has been discussed in [16].

In this paper we evaluate the two adaptive task allocation algorithms, the sensible algorithm [4] and the Random Neural Network-based Reinforcement Learning algorithm [3], that make measurement based fast online decisions to address quality of service (QoS) with low computational overhead. With no priori knowledge of workload characteristics and the state of servers in terms of resource utilization and load conditions, our approach exploits on-line measurement related to the user required QoS and make judicious allocation decisions based on the knowledge learned from the observations, adapting to changes in workload and on-going performance of the cloud environment. It is designed for the cloud service providers that use the SaaS model where the service provider sets up the VMs with the installed software components which provide the services requested by the customers.

In order to conduct experimental evaluations, we also use the Task Allocation Platform (TAP) [8], 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. In TAP, users are allowed 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. On receiving the jobs, TAP employs the accommodated allocation algorithm and dispatches the jobs to the selected machines.

Experiments are conducted on a multiple host test-bed which is heterogeneous regarding processing speed and I/O capacity, running with low to high loads that are achieved by varying arrival rates of tasks. We study the potential of our proposed algorithms when there is greater diversity both in the types of jobs, the class of QoS criteria and the SLA they request. Multi-class tasks in terms of resource requirements, such as the capacities of CPU and I/O, or agreed SLAs are injected into TAP simultaneously. We also complicate the QoS goal from minimising job execution time to economic cost of executing tasks: this cost includes the *penalty* that the cloud provider would have to pay to the end user when a SLA (service level agreement) is violated, as well as the intrinsic economic cost of using faster or slower hosts.

2 Multi-class Task allocation

It is quite common in cloud environments that the services requested by customers generate multiple workloads which are characterised differently in terms of their resource and performance requirements. For example, for an online financial management and accounting software providing SaaS services, the web browsing requests which retrieve files from web servers generate I/O bound workload, while the profit and loss accounting services which need a large amount of computation requiring high CPU capacity. Thus, the response time of the above two different web requests provided by a server relies on its capacities of the I/O and CPU respectively. TAP is designed to support multi-class tasks by assigning a distinct QoS class to a class of tasks.

2.1 Multiple QoS classes

To illustrate the multi-class task allocation, we emulate two different classes of tasks in our experiments. We first 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. The load on the I/O subsystem relies on the size of the retrieved files. In our TAP test-bed, the Apache server is deployed on each host in the cluster. The HTTPerf generates HTTP requests at a configurable rate and TAP receives the requests and dispatches them to the web servers. The second class corresponds to the web services which require a large amount of computation, mainly stress CPU and thus can be emulated by the CPU intensive job which is a "prime number generator with an upper bound on the prime number being generated". We compare the RNN based algorithms with the Sensible Algorithm, both using the goal of minimising the response time. Round-Robin scheduling is also applied as a baseline.

We conduct our experiments on a hardware test-bed as shown in Figure 1. The three hosts (with 2.8GHz, 2.4GHz, and 3.0GHz, respectively, dual-core CPU respectively) are used for task execution, while a dedicated host (2.8GHz dual-core CPU) accommodates the allocation algorithms.

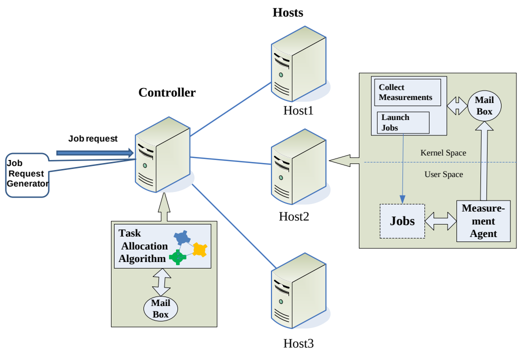
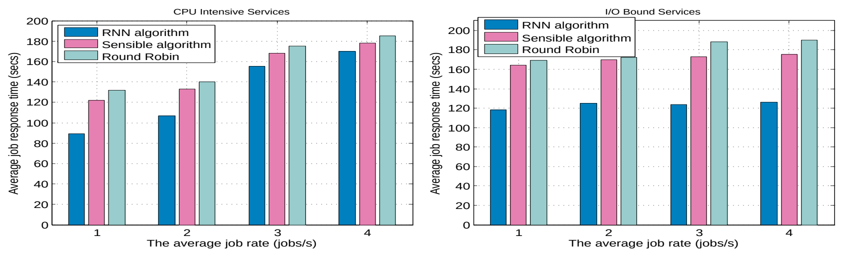


Fig. 1. The test-bed for the task allocation platform with a central controller on a test-bed

The test-bed is in the small scale as we can easily vary the load of the system and evaluate the algorithms under low, medium and high (including saturation) load conditions. The job requests arrive at TAP following a Poisson process with different average rates of $1, 2, 3, 4$ tasks/sec. To built the heterogeneous server environments, we introduce a background load which stresses I/O distinctly on each host, namely Hosts $1,\ 2,\ 3$, 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.



(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.

The results in Figure 2 show that our proposed algorithm, the RNN based algorithm and the sensible algorithm, outperform Round-Robin as they are aware of the distinct and updated performance level offered by the hosts by effective learning from the historical job experience. 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 which is heavily loaded in terms of CPU can still offer good performance to I/O bound tasks and thereby improving the resource utilization.

2.2 Contradictory QoS Requirements

In previous sections, we focus primarily on the simple QoS goal of minimising job execution/response time which is of interest of end users. From the perspective of cloud service providers, the economic cost of job execution is of greater importance. Given a certain amount of revenue of the provided services with the agreed SLA, the provider manages to use the most cost-saving machines while maintaining the requested QoS so as to improve the resource efficiency while minimising the penalty due to the violation of the SLAs.

Thus we can propose a QoS Goal which involves two contradictory requirements: if a task is dispatched to a machine with fast processors and thus higher running cost, a better response time will be offered resulting in a lower penalty due to the respect of SLAs. On the contrary the allocation of a task to a slower machine causes a lower cost for running the task, but in turn a higher possibility of being penalised due to SLA violations.

To formalise these two contradictory factors, we consider a set of host servers having distinct processing capacities, and different classes of tasks with distinct SLA agreed objects. We use I_j to represent the revenue generated by serving the tasks of a class j with the agreed SLAs. Next, we assume that the task of a class j being served by a host m which is of type M_i, where the type of host includes the specific processing capacities regarding the processor, memory and I/O, and the services offered by its software, will cause a cost to the cloud service of C_{ij}. However, the violation of SLAs will also cause some penalty to be paid by the cloud provider to the end user, resulting in the reduction of the expected revenue. For instance, there is no penalty if the response time T of the task is below the SLA agreed target value $T_{j,1} > 0$ for class j tasks. More generally, the penalty is c_{jl} if $T_{j,l-1} \leq T < kT_{j,l}$, where $T_{j0} = 0$ and $c_{j0} = 0$ (no penalty). Using the standard notation, let $1_{[X]}$ be the function that takes the value 1 if X is true, and 0 otherwise. Then from the perspective of a task of type j which is served by the cloud service, the *net income* obtained after deducing the host operating cost and the eventual penalty due to SLA violations, can be written as:

$$I_j^* = I_j - C_{ij} 1_{[m=M_i]} - \sum_{l=0}^{n} \{c_{jl} 1_{[T_{j,l} \leq T < T_{j,l+1}]}\} + c_{j,n+1} 1_{[T \geq kT_{j,n+1}]}.$$

Obviously, the cloud providers is willing to *maximise* I_j^*, while I_j is pre-defined in the service agreement.

Thus in this section we apply the task allocation algorithm with the goal of minimising the *net cost* function:

$$C_j = C_{ij} 1_{[m=M_i]} + \sum_{l=0}^{n} \{c_{jl} 1_{[T_{j,l} \leq T < T_{j,l+1}} \} + c_{j,n+1} 1_{[T \geq kT_{j,n+1}]}. \qquad (1)$$

We consider the above online financial management and accounting service as an example in the real world, the bookkeeping service requires fast response as they are frequently-used operations, while users are tolerant of the long response time for bank reconciliation service which is the time-consuming operation. It gives us an example of the tasks of two classes with the distinct SLA agreed response time and thus the distinct penalty functions.

To illustrate the preceding discussion with experiments, we use CPU intensive jobs which either are “short” with an execution time of 56ms, or “long” with an execution time of 190ms as measured on the fastest host. With regard to the first term in (1), we have $M_1 : C_{1j} = 1000$, $M_2 : C_{2j} = 2000$ and $M_3 : C_{3j} = 4000$ coinciding with our assumption that the faster machines cost more. The penalty function, which is the second term in (1), for the two distinct classes of tasks is shown in Figure 3 where the job class 2 is tolerant of longer executiontime . We emulate a heterogeneous host environment where there is a fast host, a slow host and a medium speed host by stressing the CPU of each host differently, resulting in the relative processing speeds of 1 : 2 : 4 for Host

$1,2,3$. Tasks of both classes were generated following a Poisson process with varied task arrival rates of $1,\ 2,\ 3,\ 4$ tasks/sec. The RNN based algorithm is compared with the Sensible Algorithm.

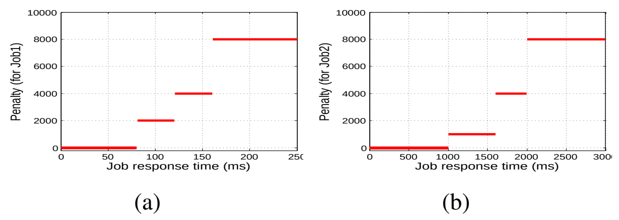


Fig. 3. The penalty function for tasks belonging to Class 1 (left) and Class 2 (right), versus the task response time on the *x* axis.

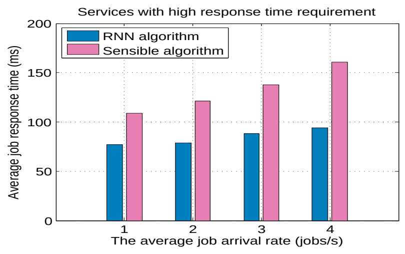
we can see that the RNN-based algorithm does better in reducing the overall cost plus SLA violation penalty (indicated as the Total Penalty in the *y*-axis) as shown in Figure 4 (c). Because the RNN is able to accurately dispatch the jobs which require fast response to the faster machines resulting in fewer SLA violations and direct the jobs which have the tolerance of long response to slower machines which can maintain the agreed SLA and thereby avoiding the interference with each other which further improves the performance.

3 Conclusions

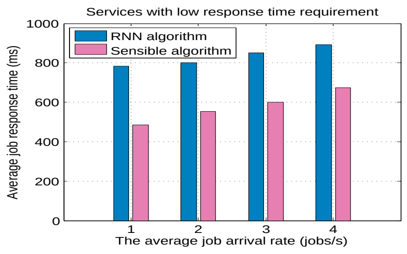
The present paper presents the design and the evaluation through experiments, of some adaptive dynamic allocation algorithms that take decisions for task allocation to servers, based on on-line and up-to-date measurements, and make fast online decisions to achieve desirable QoS level. To this effect, a Task allocation platform (TAP) is implemented as a practical system to accommodate the allocation algorithms, perform online measurement and carry out performance evaluations. Our experiments have shown the potential for enhanced performance offered by our proposed algorithms, namely the RNN-based algorithm and the Sensible algorithm, to deal with multi-class tasks. These algorithms outperform the Round-Robin scheme due to their ability to be aware of the heterogeneity in both the workload and the machine hardware, and regarding changes in load conditions in the Cloud over time. The performance of RNN based algorithm with Reinforcement-Learning stands out among the others as it offers fine-grained QoS-aware and accurate task allocation decisions.

References

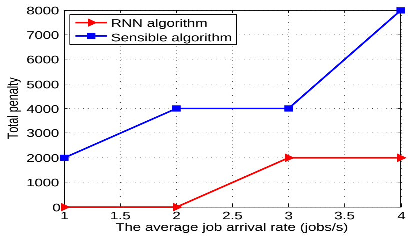
1. Chen, W., Zhang, J.: An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. Systems, Man, and Cybernetics,



(a)



(b)



(c)

Fig. 4. The average value of the measured total cost for the two classes of tasks, when allocation is based on the Goal function that includes both the economic cost and the penalty as in (1).

Part C: Applications and Reviews, IEEE Transactions on **39**(1), 29–43 (Jan 2009). https://doi.org/10.1109/TSMCC.2008.2001722

2. Delimitrou, C., Kozyrakis, C.: Qos-aware scheduling in heterogeneous datacenters with paragon. ACM Trans. Comput. Syst. **31**(4), 12:1–12:34 (Dec 2013). https://doi.org/10.1145/2556583, http://doi.acm.org/10.1145/2556583
3. Gelenbe, E., Fourneau, J.: Random neural networks with multiple classes of signals. Neural Computation **11**(4), 953–963 (1999)
4. Gelenbe, E.: Sensible decisions based on qos. Computational management science **1**(1), 1–14 (2003)
5. Gelenbe, E., Lent, R.: Trade-offs between energy and quality of service. In: Sustainable Internet and ICT for Sustainability (SustainIT), 2012. pp. 1–5. IEEE (2012)
6. Gelenbe, E., Lent, R.: Optimising server energy consumption and response time. Theoretical and Applied Informatics (4), 257–270 (Jan 2013). https://doi.org/10.2478/v10179-012-0016-1
7. Gelenbe, E., Timotheou, S., Nicholson, D.: Fast distributed near-optimum assignment of assets to tasks. The Computer Journal **53**(9), 1360–1369 (Nov 2010). https://doi.org/10.1093/comjnl/bxq010, http://dx.doi.org/10.1093/comjnl/bxq010
8. Gelenbe, E., Wang, L.: Tap: A task allocation platform for the EU FP7 PANACEA project. In: The proceedings of the EU projects track (Sep 2015)
9. Hou, E., Ansari, N., Ren, H.: A genetic algorithm for multiprocessor scheduling. Parallel and Distributed Systems, IEEE Transactions on **5**(2), 113–120 (Feb 1994). https://doi.org/10.1109/71.265940
10. Iosup, A., Ostermann, S., Yigitbasi, M., Prodan, R., Fahringer, T., Epema, D.H.J.: Performance analysis of cloud computing services for many-tasks scientific computing.

Parallel and Distributed Systems, IEEE Transactions on **22**(6), 931–945 (June 2011). https://doi.org/10.1109/TPDS.2011.66

11. Kwok, Y.K., Ahmad, I.: Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. Parallel and Distributed Systems, IEEE Transactions on **7**(5), 506–521 (May 1996). https://doi.org/10.1109/71.503776
12. Moreno, I.S., Garraghan, P., Townend, P., Xu, J.: Analysis, modeling and simulation of workload patterns in a large-scale utility cloud. Cloud Computing, IEEE Transactions on **PP**(99), 1–1 (2014). https://doi.org/10.1109/TCC.2014.2314661
13. Pandey, S., Linlin, W., Guru, S., Buyya, R.: 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. pp. 400–407 (April 2010). https://doi.org/10.1109/AINA.2010.31
14. Topcuouglu, H., Hariri, S., you Wu, M.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (Mar 2002). https://doi.org/10.1109/71.993206, http://dx.doi.org/10.1109/71.993206
15. Wang, L.: Online work distribution to clouds. In: 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). pp. 295–300 (Sept 2016). https://doi.org/10.1109/MASCOTS.2016.64
16. Wang, L., Brun, O., Gelenbe, E.: Adaptive workload distribution for local and remote clouds. In: 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC). pp. 003984–003988 (Oct 2016). https://doi.org/10.1109/SMC.2016.7844856
17. Zaman, S., Grosu, D.: A combinatorial auction-based dynamic vm provisioning and allocation in clouds. In: Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on. pp. 107–114 (Nov 2011). https://doi.org/10.1109/CloudCom.2011.24
18. Zhan, J., Wang, L., Li, X., Shi, W., Weng, C., Zhang, W., Zang, X.: Cost-aware cooperative resource provisioning for heterogeneous workloads in data centers. Computers, IEEE Transactions on **62**(11), 2155–2168 (Nov 2013). https://doi.org/10.1109/TC.2012.103
19. Zhang, Q., Zhani, M., Boutaba, R., Hellerstein, J.: Dynamic heterogeneity-aware resource provisioning in the cloud. Cloud Computing, IEEE Transactions on **2**(1), 14–28 (March 2014). https://doi.org/10.1109/TCC.2014.2306427
20. Zhuravlev, S., Blagodurov, S., Fedorova, A.: Addressing shared resource contention in multicore processors via scheduling. SIGPLAN Not. **45**(3), 129–142 (Mar 2010). https://doi.org/10.1145/1735971.1736036, http://doi.acm.org/10.1145/1735971.1736036