Improving Utilization through Dynamic VM Resource Allocation in Hybrid Cloud Environment

Yuda Wang, Renyu Yang, Tianyu Wo, Wenbo Jiang, Chunming Hu
School of Computer Science and Engineering
Beihang University
Beijing, China 100191
Email: {wangyuda, yangry, woty, jiangwb, hucm}@act.buaa.edu.cn

***Abstract*—Virtualization is one of the most fascinating techniques because it can facilitate the infrastructure management and provide isolated execution for running workloads. Despite the benefits gained from virtualization and resource sharing, improved resource utilization is still far from settled due to the dynamic resource requirements and the widely-used over-provision strategy for guaranteed QoS. Additionally, with the emerging demands for big data analytic, how to effectively manage hybrid workloads such as traditional batch task and long-running virtual machine (VM) service needs to be dealt with. In this paper, we propose a system to combine long-running VM service with typical batch workload like MapReduce. The objectives are to improve the holistic cluster utilization through dynamic resource adjustment mechanism for VM without violating other batch workload executions. Furthermore, VM migration is utilized to ensure high availability and avoid potential performance degradation. The experimental results reveal that the dynamically allocated memory is close to the real usage with only 10% estimation margin, and the performance impact on VM and MapReduce jobs are both within 1%. Additionally, at most 50% increment of resource utilization could be achieved. We believe that these findings are in the right direction to solving workload consolidation issues in hybrid computing environments.**

***Keywords*—Hybrid Cloud Environment; VM Resource Dynamic Allocation; VM Migration; MapReduce**

I. Introduction

We are experiencing a *Big Data* era. A study by Harvard Business Review in 2012 shows that 2.5 Exabyte of data is generated everyday and the speed of data generation doubles every 40 months [16]. To satisfy the increasing demands of data analysis and processing, several systems from both academia and industry are proposed to perform the compute-intensive tasks among distributed nodes such as Mesos [14], Yarn [23], Fuxi [26] etc. Meanwhile, virtualization has been evolved into one of most important techniques in cloud computing in recent years. By leveraging virtualization, the data center providers could not only reduce the maintenance cost of the infrastructure, but also increase the elasticity and energy-efficiency through on-demand resource allocation and workload consolidations. Multi-tenant customers are thus capable of sharing the underlying resources and running a variety of transactional or long-running services such as web service, virtual desktop etc. in their isolated environment. All these diverse workloads constitutes elements within the hybrid cloud environment. However, service providers still need to address a number of key challenges, such as striking a balance between the application performance and the cluster resource utilization. In fact, the recent workload study in cloud datacenter illustrates that the low utilization is a very normal phenomenon. As illustrated in [19], only 53% memory and at most 40% CPU are utilized respectively on average.

One solution for utilization improvement is to run separate frameworks on the same cluster, intuitively sharing the same physical resources. At certain point of time, different workloads (e.g., memory-intensive task and CPU-intensive task) might utilize different resources complementarily. Nevertheless, individual frameworks lack of overall cluster-level resource viewpoints, resulting in non-negligible overhead in data exchanges and component communications. To further improve the cluster utilization, combinations of different types of workloads are taken into account in an uniformed framework. Due to the diverse resource requirements of the heterogeneous workloads, the complicity of resource management and scheduling would dramatically increase with inefficient request handling. Both Yarn and Fuxi adopt similar architecture to support different computation paradigms such as MapReduce [11], DAG paradigm [1], streaming [6], Spark [25] etc. However, they have not yet considered how to merge virtual machine (VM) workload within the current resource management system. Unlike those tasks, VM is long-running service and tends to be infrequently created and moved once being initially placed to a specific node. In fact, VM would be migrated only when the system failure and node overload are detected due to the substantial costs with many factors considered [15] during the migration.

In cloud environment, VMs are usually configured during the initialization with the amount of resource (CPU, memory) specified [17]. Resource over-provision is one of the solutions to lower the SLA violation from users point of view but usually leads to poor infrastructure utilization. In contrast, under-provision could lead to potential performance degradation despite the increased utilization. Additionally, the pre-defined size of VMs and vertical application elasticity would also result in the decrement of resource utilization [22]. Therefore, there are a great number of opportunities for the providers to dynamically adjust the actual resource allocated whilst guaranteeing the QoS of applications inside the VM.

In this paper, we propose a workload consolidation approach which combines traditional batch processing tasks (e.g, MapReduce) with online long-running VM workload in a hybrid computing environment. We design an elastic resource allocation mechanism to timely adjust the amount of the over-provision according to the real application resource usage with

little negative impact on other co-allocated computing tasks. Our system is constructed based on Yarn and KVM, and the experimental results reveal that the allocated memory with the dynamic allocation approach is close to the real usage with only 10% estimation margin. Additionally, the performance impact on VM and MapReduce jobs performance are both within 1% while achieving at most 50% utilization increase. In particular, the major contributions of our work can be summarized as follows:

- A new protocol that efficiently handles user's multiple VM placement requests to mitigate the unbalanced resource aggregation.
- A sliding window based VM resource adjustment mechanism in order to improve the holistic cluster utilization.
- A VM-migration based approach that enables the high availability and performance for the VM application.

The remaining sections are structured as follows: Section II presents the background and basic workflow while Section III describes the proposed approaches and core design philosophy; Section IV shows the experimental results followed by the related work in Section V; Finally, we conclude our paper in Section VI with the future work discussed.

II. System Overview

A. Background

To satisfy the increasing demands of diverse data processing framework, two-level scheduling architecture (such as Yarn, Fuxi) is widely adopted to de-couple the individual programming models from the resource management. Specifically, Resource Manager (RM) is the central resource negotiator and responses to different application's resource requests whilst granting available resources to the Application Master (AM). AM coordinates the logical plan for each computation tasks and requests resources from the RM. Upon receiving the granted resource, AM would interact with the Node Manager (NM), a special system worker daemon to launch, monitor or kill the compute task within an execution container. The container could be regarded as a logical bundle of resources bound to a particular node and named as *resource release*.

In hybrid cloud environments, the resource management is becoming more complicated with the combinations of different heterogenous workloads or frameworks. On a shared cluster, the compute-intensive job (e.g., MapReduce), web service, long-running VM workloads etc. are co-allocated. Based on the core philosophy in Yarn, we design and implement a protocol to support both MapReduce application and VM placement.

B. Basic Workflow

In our system, we inherit some basic concepts from Yarn. Each master (e.g., MR Master and VM master) is responsible for application-specific execution logic and the corresponding resource request. The workflow in Figure 1 illustrates the lifecycle of a job and how system components cooperate with each other. Firstly, VM requests are submitted to the RM. Once available resources are detected in RM scheduler, RM will allocate a container for the VM Master (step 1-3).

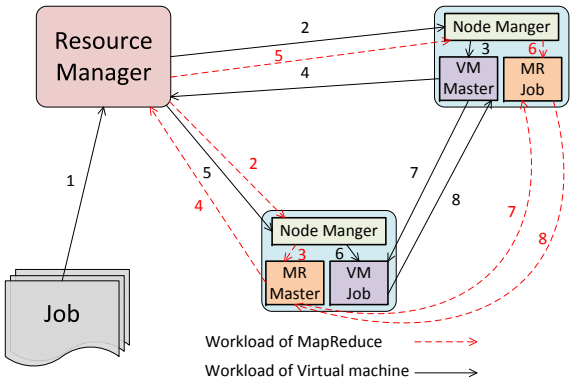


Fig. 1: The workflow of hybrid computing environment

Afterwards, the VM launching request would be handled by VM master and a new protocol for VM placement is adopted with a global cluster resource view considered. VM Master then delivers the resource request to the RM (step 4). Each request contains resource demands and the locality preference generated by the protocol. RM will attempt to satisfy the requests, and VM Master will send a concrete work plan to the corresponding NM (step 5-6) to launch a VM upon receiving assigned resources. Thereafter, NM starts the VM in a resource container, which is limited and isolated by Linux Cgroups [2]. The VM reports its running state to VM Master via heartbeat (step 7-8) periodically.

In addition, we utilize a dynamic resource allocation mechanism to adjust the assigned resource amount according to the collected profiling information produced by NMs. Furthermore, an automated migration might be incurred by VM master to guarantee the high availability and performance of running workloads. When the migration terminates, there will be a free container and the VM Master will send to NM a decreased container plan, to subsequently kill the container and reclaim the corresponding resources to RM.

III. Design and Solutions

In order to fulfill the scenario mentioned above, the system architecture is proposed as demonstrated in Figure 2. In RM, the node tracker traces the running status, and RM maintains a waiting queue for job requests and handles resource requests. Based on the Application Master service, VM Master controls VMs with a placement protocol and takes responsibility for the VM routine execution and migration. In addition, a dynamic allocator in each node takes charge of dynamic allocation and VM migration. More details will be discussed in the following subsections.

A. A New Protocol for VM Placement

In MR Master, data locality and load balance are usually taken into account during scheduling. For example, the map task in MapReduce would be assigned to the node that is close to the data block. However, interactive latency is the dominant factor considered by VM service, indicating that the VM job

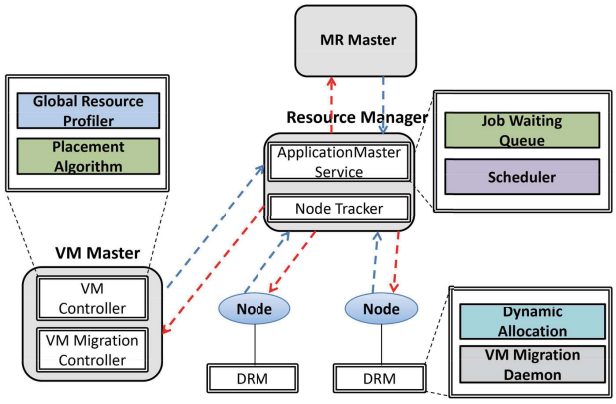


Fig. 2: System Architecture

should be placed on a node with as sufficient resources as possible. Our protocol is thus designed for the VM placement.

Firstly, VM Master delivers a *FullInfoRequest* to RM to obtain the overall resource information when the master process is created and the priority segment is also carried within the message. Meanwhile, RM maintains an unique request waiting queue for the multiple requests, following the first come and first service (FCFS) rule. Besides, preemption mechanism is introduced so that applications with the lowest priority could be preempted to make space for higher ones. RM aggregates a global resource view collected from all NMs, and it composes the required information into *protobuf* [5] format to respond to corresponding VM Master once the request is satisfied.

Secondly, informed the available resources of each node, VM Master determines which node the job should be assigned to by using an extended load-balanced algorithm (See Algorithm 1). The overall resource profile is traversed firstly and the node with the maximum available resources is sorted out. If multiple nodes are found, other workloads running on each node should be checked and the node with the minimal number of MR jobs would be finally selected. We also implement a plugin interface to support multiple VM placement strategies such as [13], [18] etc.

Furthermore, VM Master encapsulates resources demands (including the initial resource need of the VM, the placement instruction generated by the heuristic algorithm and the priority of the request) into a *ResourceRequest* which will be delivered to RM asking for execution containers.

All these processes constitute an atomic operation due to the resource contention and interference from co-allocated VMs [24]. Therefore, the priority queue should be locked when *FullInfoRequest* is being handled and be unlocked after a VM manages to be launched in a container. Obviously, the locking mechanism limit the concurrency of VM requests and the system scalability. This inspires us to conduct further optimizations.

Figure 3 could be demonstrated as an example. If a user submits three VM requests, the consumed time of sequential VM placements is 1.8 times more than merged request. The performance degradation becomes even more serious with the

Algorithm 1 VM placement pseudo-code

```
1:  NodeResourceList ← Obtain the overall resource profile
2:  VMPlacementList ← null, MaxResource ← 0
3:  for each NodeResource in NodeResourceList do
4:      if NodeResource > MaxResource then
5:          MaxResource ← NodeResource
6:          clear VMPlacementList
7:          VMPlacementList ← NodeResource
8:      else if NodeResource = MaxResource then
9:          add NodeResource to VMPlacementList
10:     end if
11: end for
12: if |VMPlacementList| = 1 then
13:     place VM on this node
14: else
15:     traverse VMPlacementList, find the node with the least
    MR jobs executing on
16:     place VM on this node.
17: end if
```

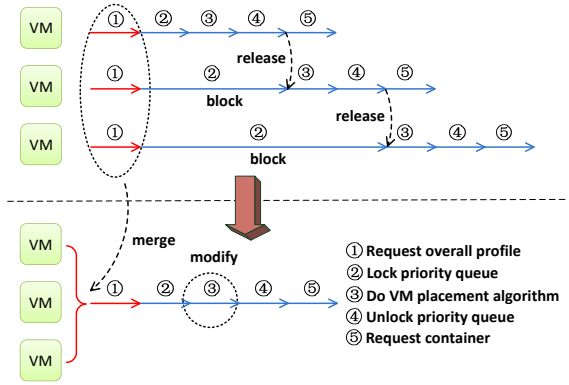


Fig. 3: Merge of VM requests

increment of VM request number. Other VMs have to wait and be blocked until the locking is removed by the VM which obtained the lock. In fact, due to the homogeneity of the same user requirement, both the full information synchronization (step 1) and the acquisition/release of the shared lock (step 2 and 4) could be merged together. More specifically, the pre-fetched overall resource information resides into local memory and the queue is locked. Afterward, the proposed placement will be conducted until all VMs generate their optimal location preferences. During these periods, the global resource profile is accordingly modified when each round finishes. All VMs' requests are combined into a single message after releasing the lock, resulting in significant reduction of communication overheads.

B. Dynamic Resource Allocation for VM

The long-term resources occupancy of VM service and the constant resource leases in a node would all lead to low cluster resource utilization. Thus, we design a dynamic resource allocation mechanism for VM to handle with this issue.

After RM allocates the lease on a particular node, VM

Algorithm 2 Sliding Window Based Dynamic Allocation

Define:

currentAllocatedResource ← currently isolated resource for VM job
availableResource ← the available resource of the node
retryTimes ← pre-defined size of sliding window
left ← the left of sliding window
assignedResource ← the value of resource to be isolated time-slot Γ
Monitor the resource occupation of VM job M(Γ)

```
1:  while Γ do
2:      if the elements number in the window < retryTimes then
3:          add M(Γ) to sliding window
4:          continue
5:      else
6:          if all element values in window are same then
7:              assignedResource ← the window value
8:              move left to window end
9:          else
10:             left ← left + 1
11:             continue
12:         end if
13:     end if
14:     if assignedResource < currentAllocatedResource OR
        availableResource is sufficient then
15:         currentAllocatedResource ← assignedResource
16:         modify AvailableResource
17:         inform RM
18:     else
19:         do Migration
20:     end if
21: end while
```

Master will launch a VM based on this container (a resource lease). A daemon running on this node monitors the real-time resource usage of the VM through Linux *proc*, which is a pseudo file system providing an interface to get access to the runtime system and process information. Based on this information, a dynamic resource allocation algorithm is implemented to determine the new size of the container. Since the amount of the isolated resource is computed with over-provision, there are substantial spaces to shrink the amount of the binding resource. The frequency of the lease adjustment is another intricate challenge. Frequent re-allocation would aggravate the burden of resource scheduler and increase the network load as well when exchanging too much information with RM. However, long period modulation, on the other hand, would negatively impact the allocation effect and fail to improve the server utilization.

To strike the balance between them, we sacrifice some computation accuracy by quantifying the allocated resource amount to each VM into an integer number. We use a *sliding window* based mechanism to cope with the transient spike, avoiding unnecessarily repetitive re-allocations. If the assigned resource changes and remains stable in the next time period of the window size, there is a potential trend for the new resource re-allocation. In practice, we use the *retryTimes* as the pre-defined size of the *sliding window* and once all sequential values inside the window approximately keep the same, the currently assigned resource could be substituted by the new *sliding window* value. The core idea in Algorithm 2 is presented .

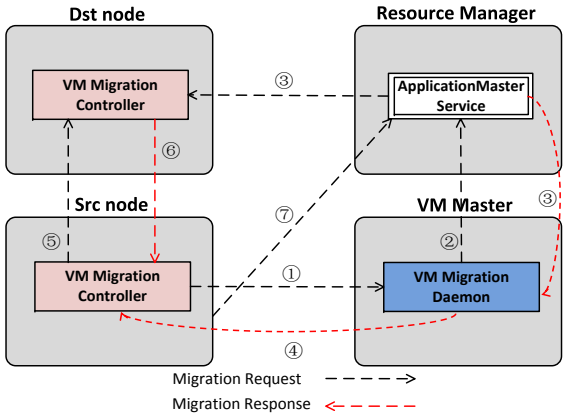


Fig. 4: The process of migration

There will be three different types of situation in terms of the dynamic re-allocation. If the new resource size is less than the current allocated resource, the assigned amount will be cut down. Otherwise, if the new resource exceeds the current allocation while the node has sufficient resource to offer, the allocation will be subsequently conducted. If the lease expansion overweighs the physical nodes capability, some workloads migration will be carried out to make enough spaces for execution.

If the current node is able to afford the resource scaling up or down, the dynamic allocation approach would generate a new resource assignment plan directly. It firstly notifies the NM to re-isolate the container, followed by communicating with RM to update the global resource view using the newly assigned amount. In the protocol, new message segment is added with the original heartbeat between NM and RM, which is typically used to report the node states. Upon getting the heartbeat from NM, RM will initially check the new segment. If the segment is not empty, RM extracts the package and updates the global resource profile.

C. Migration

A VM migration daemon is running along with the NM and it makes preparation for the migration, sends/receives instructions and performs migration as well. To fulfill live migration, the resource lease on the destination node must be allocated beforehand to reduce the service downtime. VM migration controller (VMMC) in VM Master is responsible for this type of resource allocation.

The whole process is illustrated in Figure 4. Once the daemon in source node captures the migration instruction, it generates a *MigrationRequest* with needed resources in destination node and sends the request to VM Master (step 1). When receiving the *MigrationRequest*, VMMC of the VM master will ask the RM for a particular node using our proposed VM placement protocol. RM allocates the resource,

creates the lease on the specific node and responds to the VM Master with the appointed assignment (step 2-3). Then VM Master will inform the source node about the destination address. Consequently, the VMMC in the source node gives the migration command to move the VM to the destination node (step 4-5). After receiving *MigrationFinish* signal, VMMC from the source will release the used resources which will be recruited by RM later (step 6-7).

IV. Experiments and Evaluation

In this section, we discuss and explain the effectiveness of the proposed dynamic resource allocating for VM in hybrid computing environment and then evaluate the efficiency in improving the cluster resource utilization.

Before explaining the experiments, we will introduce the benchmarks consisting of transactional workloads which all run inside the VM:

Tpc-c [7]: an on-line transaction processing benchmark, which simulates a complete computing environment where a large number of users execute transactions against a specific database. These transactions include: entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses.

Wikipedia [8]: also an online transaction processing benchmark, which executes adding and removing watch list, updating page, getting page anonymous or authenticated. Wikipedia emphasizes more on memory data operations.

Memcached [3] [4]: a general-purpose memory caching system, which is often used to speed up dynamic database-driven websites by caching data and objects in RAM or used to test memory latency.

Moreover, we use WordCount for MR workload, and the Job Completion Time (JCT) is calculated to measure the impact on the job performance.

A. Experimental Environment

We use 5 machines to constitute a distributed computing environment in which MR jobs and VMs could be co-executed. Our system is extended from current Yarn 2.2.0 and the VM service is based on QEMU-KVM (kvm-84). Each machine consists of Intel(R) Core(TM) i7 860@2.80GHz CPU, 4GB memory and 320GB disk.

B. Experimental Results

1) ***Dynamic Resource Allocation Effect****:* To demonstrate the improved resource utilization through dynamic resource resizing, we firstly demonstrate the effect of proposed resource binding mechanism under different types of workload mixtures. Afterwards, we evaluate the performance impact on the utilized benchmark workload inside the virtual machines over a period of time.

Firstly, we set up a pair of VMs on each node in our test cluster with 768M memory initially required for each VM. Diverse workloads combinations are made over the cluster as shown in Figure 5 and each benchmark runs in a VM. More specifically, TPC-C simulates 20 concurrent users while

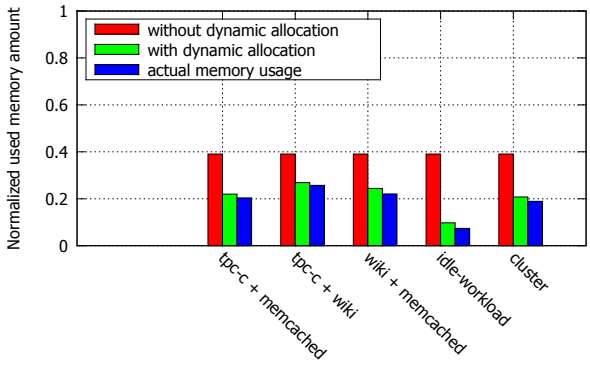


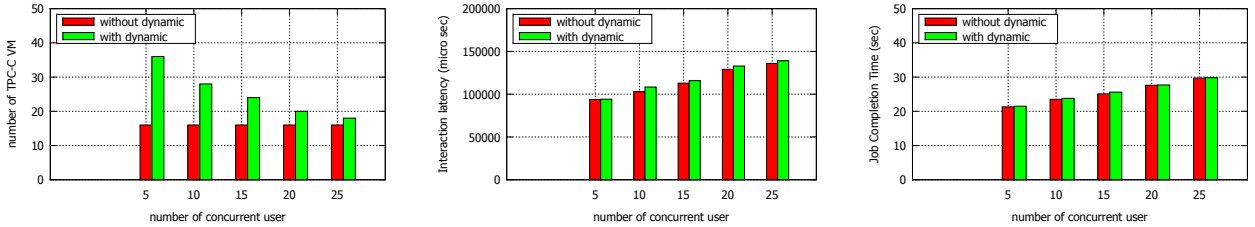
Fig. 5: The normalized memory usage of each node and the whole cluster. Pair VMs with different workloads are executed on each node over the cluster. The figure shows the assigned memory amount with, without dynamic allocation mechanism and the actual memory usage are compared respectively.

Wikipedia simulates 10 concurrent users with a 1M dataset. Each benchmark simulates 1000 transactions per second. We record the assigned and the actual used memory amount after workloads are launched and could be executed stably. Apparently, the assigned memory without dynamic allocation(static allocation) approximately is twice the actual used amount, resulting in a fact that substantial allocated resources are occupied and would not be highly utilized. In contrast, the allocated memory with our proposed mechanism is extremely close to the usage with only 9.89% margin on average in the whole cluster. A magnified phenomenon can be observed in idle-workload node because the required resource to run VMs is far from the initial request amount configured when launching the VM. We also believe that our method will significantly improve the resource utilization especially in idle workloads environment.

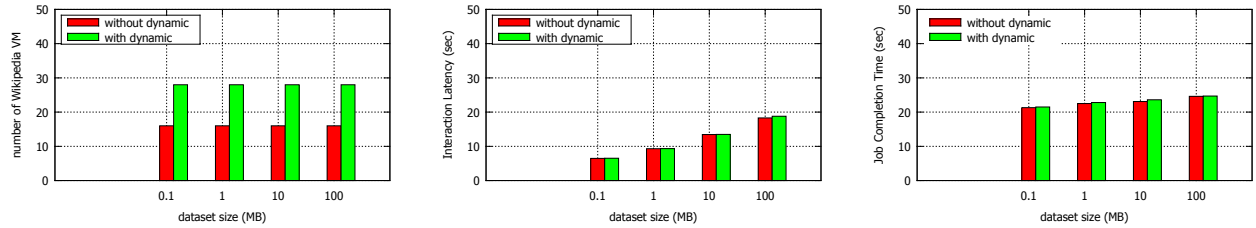
Furthermore, we launch a VM with the same resource request and execute the TPC-C benchmark simulating 20 concurrent users inside the VM. Similarly, we record the average assigned resource with/without dynamic mechanism and the actual memory consumption every 5 seconds when the VM is launching. With the VM initialized, the real memory usage is growing. The static method assigns 800M memory over the whole time duration and the dynamic resource allocation is also illustrated in Figure 6(a). In addition, the interaction latency of the benchmark is also monitored and the result shown in 6(b) reveals that the average latency is slightly increased by at most 1% which can be nearly neglected.

2) ***Impact on Performance of VM workload and MR job****:* Previous experiments have demonstrated the dynamic allocation mechanism could spare and reuse the resources which are assigned out but not really utilized by the VM. We further illustrate the performance impact on different workloads in our hybrid computing environments.

In our test cluster, we keep five wordcount jobs concurrently running by continuously launching a new wordcount job once one job finishes. At the mean time, we constantly submit VM with TPC-C benchmark simulating 5 concurrent user and 1,000 transactions per second until no more VM could

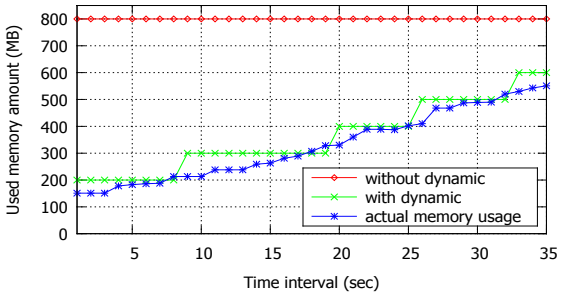


(a) the number of TPC-C VM (b) the performance of TPC-C VM (c) the average JCT of wordcount jobs

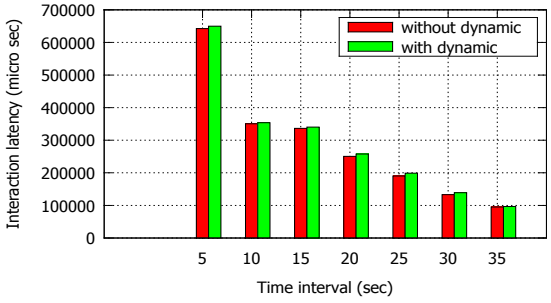


(d) the number of Wikipedia VM (e) the performance of Wikipedia VM (f) the average JCT of wordcount jobs

Fig. 7: The maximum number of VM that can be supported, the VM workload interaction latency and the JCT of the co-executed wordcount jobs under different TPC-C and Wikipedia workload stress.



(a) Used memory amount (MB)



(b) TPC-C interaction latency

Fig. 6: In (a), The actual memory usage, dynamic and static resource allocation in each time interval; The corresponding TPC-C workload interaction latency in (b).

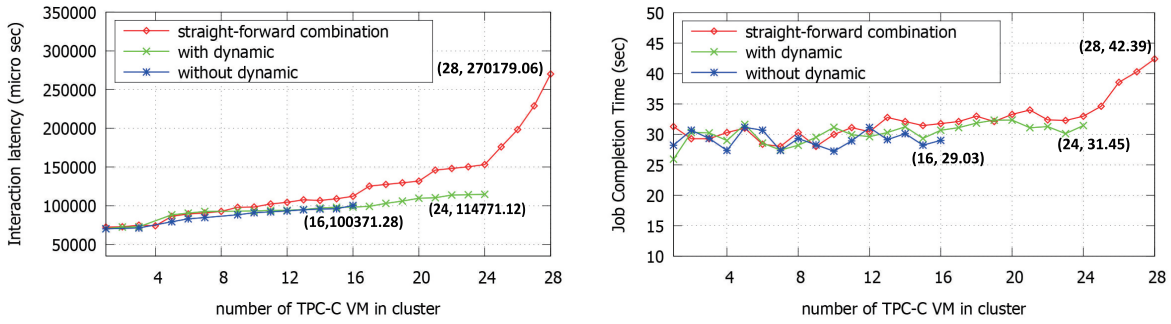
be launched and scheduled. The whole experiment lasts for 600 seconds. In addition, we measure the maximum number of VMs that the system can support, the VM's interaction latency, and the average JCT of all wordcount jobs respectively. Furthermore, we repeat similar experiments by adjusting the parameters of TPC-C benchmark (with 10, 15, 20, 25 concurrent users respectively) and by adopting Wikipedia benchmark (with 0.1M, 1M, 10M, 100M dataset respectively) to further evaluate the impact on the performance under different workload conditions. Figure 7 depicts the results of above experiments.

As shown in Figure 7(a) and 7(d), with the increment of the simulated user number in TPC-C workload, the stress of each VM workload grows. The supported VM number by our proposed mechanism thus decreases due to the fixed cluster resource amount. On the contrary, the number in the static allocation approach remains the same, much less than our dynamic allocation mechanism. This is because the overall memory is not fully utilized even though some memory can be spared by co-allocated running VMs. Since Wikipedia focuses on memory data operations, weak performance interference among VMs leads to a similar VM allocation in all cases. All in all, by leveraging the dynamic resource strategy, more VMs can be performed on the cluster and the resource utilization is consequently highly improved.

Figure 7(b) and 7(e) illustrate the workload performance comparison under different configurations. There is only a slight overhead (within 1%) in comparison with the static allocation strategy. For example, the latency of TPC-C with 10 concurrent users increases from 23.5 microseconds to 23.8 microseconds. This minor overhead can be almost ignored considering the significant improved utilization.

It is also observable that the extended JCT of word count workload is no more than 1%. Specifically, only 0.1s is needed to finish the word count job when 100M dataset is adopted in Wikipedia workload. The results in all cases reveal that there is little impact on the performance of co-executed batch workloads. we attribute it to the isolation efficiency of our platform.

*3) **Performance Comparisons Among Different Approaches**:* We conduct the experiment to compare the performance of our approach with straight-forward combina-



(a) The maximum number of supported VM and the response latency of different methods

(b) WordCount job complete time

Fig. 8: Performance impact comparison among different approaches

tion of VMs and MapReduce framework in one cluster. We firstly make five wordcount jobs concurrently running by continuously launching a new wordcount job once one job finishes. At the same time, VM is launched in our system with dynamic resource allocation and static allocation scheme. For comparison, we also startup VM by primitive KVM manager. The interactive latency of VM and JCT of wordcount within different stage are recorded and depicted in Figure 8.

Due to the resource hard limit set by static resource allocation, the maximum number of VM that can be set up is only 16. No more VMs can be started up, even if some spared memory could be shared or temporarily taken over. The proposed dynamic allocation strategy brings considerable benefits by tightly allocating the requisite resources so that 8 more VMs could be launched. Meanwhile, the response latency (shown in Figure 8(a), from 100,371.28 microseconds to 114,771.12 microseconds) and the corresponding JCT (shown in Figure 8(b), from 29.03 seconds to 31.45 seconds) is slightly increased. From the providers' viewpoint, this approach will be, no doubt, worthy of sacrificing little overhead for the greatly improved cluster utilization.

Figure 8 also demonstrates that the performance of both workloads dramatically descend although much more VMs could be launched by primitive KVM commands. Compared with dynamic resource allocation, the interactive latency of TPC-C workload in straight-forward combination dramatically increases by 135% from 114,771.12 microseconds to 270,179.06 microseconds (Figure 8(a)), and the JCT of wordcount increases by 34.78% (Figure 8(b)). The reason for this is that the naive scheme lacks hard resource limit to resource and an effective isolation strategy, bringing about intense resource competition among co-allocated workloads. In fact, the increased latency can not be tolerated due to severe SLA violation, and is thus not an ideal solution in a hybrid computing environment.

*4) **Performance Benefits from VM migration**:* In order to reduce the SLA violation, we perform VM migration when potential system overload is detected. In this experiment, we generate an overloaded scenario by artificially aggravating existing workloads and make the available memory of the server below 100M. Afterwards, we increase the inner workload of a specific VM and measure the fluctuation of the interaction latency. According to our approach, the VM is supposed to be migrated once it cannot obtain sufficient resources from its current node. It is observable from Table I that the latency of Wikipedia and TPC-C benchmark are reduced by 32.45% and 18.84% respectively through VM migration, thereby significantly maintaining the user's QoS.

TABLE I: the interaction latency comparison before and after VM migration

Workload	Before migration	After migration	reduction
Wikipedia	6467078	4368376	32.45%
TPC-C	86094.29	69876.83	18.84%

V. Related Work

This section describes and discusses the related work towards handling problems in hybrid computing framework and resource provisioning for virtual machine.

Hybrid computing framework with MapReduce: One issue in a cluster monopolized by an individual computing framework is the low resource utilization [19]. To solve this problem, combining different types of workloads in an unified framework becomes an option. Some recent solutions are Yarn [23], Mesos [14], Fuxi [26], Corona [9], Omega [20], which are maintained and used respectively by Apache, Twitter, Alibaba, Facebook and Google. Most works separate the functionality of job execution from the resource management. Job execution only takes responsibility for the lifecycle of a specific job while resource management provides an unified resource scheduling and assignment. However, current approaches only focus on the combinations between batch processing and real-time computing jobs but neglect long-running services. Moreover, the static resource lease mechanism limits the flexibility of resource allocation. Sharma et al. [21] introduce a model to hybrid MapReduce with VM. The model divides the cluster into a virtual machine sub-cluster and a physical sub-cluster. Submitted interactive jobs will be placed on virtual cluster and the deployment of MapReduce will be determined by the overhead of executing on virtual machine through a small training cluster. Nevertheless, the model would influence the job completion time of MapReduce to some extent.

Resource over-provision for virtual machine: Resource over-provision plays a key role in ensuring that the cloud

providers adequately accomplish their obligations to customers while maximizing the utilization of underlying infrastructure. Typically, there are two kinds of over-provision, static resource over-provision and dynamic resource provisioning. Static provisioning is often applied in the initial stage of capacity planning. It is usually conducted in offline and occurs on monthly or seasonal timescales. The pre-allocated resource is an estimated size based on its workload pattern. However, workload fluctuations and overall low-utilized servers at most time all lead to substantial resource waste during non-peak times. There are also studies concentrated on dynamic resource provisioning. Ghosh et al. [12], analyse the risks of over-provision resources in a cloud and a threshold-based scheme is proposed. Similarly, Breitgand et al. [10] present an algorithmic framework to estimate the total physical capacity required to perform the over-provision. They rely on the insight in future resource usage. In contrast, we provide VM service for customers, which can hardly predict the user's operation of the next period. In addition, those approaches consider each VMs resource need separately. With multiple fixed jobs (MR and VM) scheduling and executing in hybrid computing cluster, such frequent adjustment in real-time and fine-grained way might result in the available resources of the cluster fluctuating so rapid that the resource scheduling could be imprecise.

VI. Conclusions and Future Work

In this paper, we introduce a hybrid computing environment to combine long-running VM service with traditional batch workloads. Firstly, a new protocol is proposed to deal with the VM placement problem and we conduct optimizations to reduce the serial requests waiting time. Secondly, we build a dynamic resource allocation mechanism for VM in order to improve the holistic cluster utilization. Additionally, VM migration is utilized to ensure HA and avoid the potential performance degradation. The experimental results reveal that the dynamically allocated memory is close to the real usage with only 10% estimation margin, and the performance impact on VM and MapReduce jobs are both within 1%. Resource utilization with our approach could also reach at most 50% increase. Our current work concentrates more on memory isolation and re-allocation. As for the future work, we believe that on-demand CPU control and adjustment should be enhanced for CPU intensive workloads. Furthermore, the VM failover is also needed to be included in the resource management component.

Acknowledgements

We thank the reviewers for their helpful feedbacks. The work in this paper is supported by National Basic Research Program of China (No.2011CB302602), China 863 Program (No.2013AA01A213) and NSFC (No.91118008, 61170294).

References

[1] Apache Tez. http://hortonworks.com/hadoop/tez/.

[2] Linux Cgroups. http://en.wikipedia.org/wiki/Cgroups.

[3] mcblaster Benchmark. https://github.com/fbmarc/facebook-memcached-old/tree/master/test/mcblaster.

[4] memcached Benchmark. http://memcached.org/.

[5] Protocol Buffers. http://en.wikipedia.org/wiki/ProtocolBuffers.

[6] Storm: Distributed and fault-tolerant realtime computation.

[7] TPC-C Benchmark. http://www.tpc.org/.

[8] Wikipedia Benchmark. oltpbenchmark.com/wiki/index.php?title=Workloads.

[9] Under the Hood: Scheduling MapReduce jobs more efficiently with Corona. http://on.fb.me/TxUsYN, 2012.

[10] D. Breitgand, Z. Dubitzky, A. Epstein, A. Glikson, and I. Shapira. Sla-aware resource over-commit in an iaas cloud. In *Proceedings of the 8th International Conference on Network and Service Management*, pages 73–81. International Federation for Information Processing, 2012.

[11] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1), 2008.

[12] R. Ghosh and V. K. Naik. Biting off safely more than you can chew: Predictive analytics for resource over-commit in iaas cloud. In *5th International Conference on Cloud Computing (CLOUD)*, pages 25–32. IEEE, 2012.

[13] F. Hermenier, X. Lorca, J. marc Menaud, G. Muller, and J. L. Lawall. Entropy: a consolidation manager for clusters. In *International Conference on Virtual Execution Environments(VEE)*, pages 41–50. ACM, 2009.

[14] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation(NSDI)*, pages 22–22. Usenix, 2011.

[15] J. L. Lucas-Simarro, R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Scheduling strategies for optimal service deployment across multiple clouds. *Future Generation Computer Systems*, 29(6):1431–1441, 2013.

[16] A. McAfee and B. Erik. Big data: The management revolution. *Harvard Business Review*, 10 2012.

[17] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis. Efficient resource provisioning in compute clouds via vm multiplexing. In *Proceedings of the 7th International Conference on Autonomic Computing(ICAC)*, pages 11–20. ACM, 2010.

[18] I. S. Moreno, R. Yang, J. Xu, and T. Wo. Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement. In *2013 IEEE 11th International Symposium on Autonomous Decentralized Systems ISADS (ISADS)*, volume 1, pages 1–8. IEEE, 2013.

[19] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing(SoCC)*, page 7. ACM, 2012.

[20] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems(EuroSys)*, pages 351–364. ACM, 2013.

[21] B. Sharma, T. Wood, and C. R. Das. Hybridmr: A hierarchical mapreduce scheduler for hybrid data centers. In *IEEE 33rd International Conference on Distributed Computing Systems(ICDCS)*, pages 102–111. IEEE, 2013.

[22] L. Tomás and J. Tordsson. Improving cloud infrastructure utilization through overbooking. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, page 5. ACM, 2013.

[23] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing(SoCC)*, page 5. ACM, 2013.

[24] R. Yang, I. S. Moreno, J. Xu, and T. Wo. An analysis of performance interference effects on energy-efficiency of virtualized cloud environments. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, volume 1, pages 112–119. IEEE, 2013.

[25] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.

[26] Z. Zhang, C. Li, Y. Tao, R. Yang, H. Tang, and J. Xu. Fuxi: Fault tolerant resource management and job scheduling system at internet scale. In *Proceedings of the 40th International Conference on Very Large Data Base(VLDB)*, 2014.