Tetris: Optimizing Cloud Resource Usage Unbalance with Elastic VM

Xiao Ling*, Yi Yuan†, Dan Wang†, Jiahai Yang*
*Institute for Network Sciences and Cyberspace, Tsinghua University
†Department of Computing, The Hong Kong Polytechnic University
lxcernet@gmail.com, {csyiyuan, csdwang}@comp.polyu.edu.hk, yang@cernet.edu.cn

***Abstract*—Recently, the cloud systems face an increasing number of big data applications. It becomes an important issue for the cloud providers to allocate resources so as to accommodate as many of these big data applications as possible. In current cloud service, e.g., Amazon EMR, a job runs on a fixed cluster. This means that a fixed amount of resources (e.g. CPU, memory) is allocated to the life cycle of this job. We observe that the resources are inefficiently used in such services because of resources usage unbalance. Therefore, we propose a runtime elastic VM approach where the cloud system can increase or decrease the number of CPUs at different time periods for the jobs. There is little change to such services as Amazon EMR, yet the cloud system can accommodate many more jobs. In this paper, we first present a measurement study to show the feasibility and the quantitative impact of adjusting VM configurations dynamically. We then model the task and job completion time of big data applications, which are used for elastic VM adjustment decisions. We validate our models through experiments. We present Tetris, an elastic VM strategy based on cloud system that can better optimize resource utilization to support big data applications. We further implement a Tetris prototype and comprehensively evaluate Tetris on a real private cloud platform using Facebook trace and Wikipedia dataset. We observe that with Tetris, the cloud system can accommodate 31.3% more jobs.**

I. Introduction

By offering elastic computing and storage resources, cloud computing is changing the landscape of enterprise's computing infrastructure. Cloud providers build data centers with a huge amount of machines. Using virtualization technique, the cloud can provide *virtual machines* (VMs) to the end-users in a pay-as-you-go manner. From the cloud providers' point of view, a critical problem is to allocate its resources (CPU, network, etc.) effectively so as to accommodate more user jobs.

Recently, there is an increasing number of big data applications. The de facto standard to process massive data is MapReduce [1]. For grass root users or non-computing professionals, the cost for deploying and maintaining a large-scale dedicated server clusters can be prohibitively high, not to mention the technical skills involved. They naturally resort to the cloud systems. As such, the cloud providers see increasing demands in cloud-based big data applications.

There are several approaches that current cloud platforms support big data applications. The first representative approach is Google App Engine-MapReduce [2]. It is a pure *Platform As A Service (PaaS)* approach where users directly submit their MapReduce jobs to the Google App Engine platform. The advantage of this approach is that the users do not need to worry about the details, yet they also yield the control of their applications to the cloud. The second is represented by Amazon EMR [3] and Microsoft HDInsight [4]. A user first provisions VMs (Amazon EC2 or Microsoft Azure) from the cloud provider to construct a cluster. The cloud provider then deploys a big data processing system such as Hadoop, Spark [5] on top of the cluster for the user. The user application is then executed. The users are charged by the consumed infrastructure with certain additional cost of maintaining the system. The advantage of this approach is that it gives the users a full control of the system. Users own their cluster (i.e., do not need to share with others) and it is easy for the users to switch their applications to another cloud provider if needed. We call this kind of approach as *infrastructure-based PaaS (I-PaaS)*. This approach is widely adopted and it is our focus in this paper.

Similar to a standard IaaS platform, e.g., Amazon EC2, in such I-PaaS, each of the VMs is allocated with a fixed amount of resources (e.g., CPU, memory, disk I/O and network bandwidth). We observe that when executing MapReduce jobs, the amount of resources required at different times in the MapReduce job life cycle differs. Specifically, big data applications are not only CPU intensive; but other resources, e.g., the I/Os, may also become bottleneck at different time period. In such situations, either CPU or the I/O will be idle; and a fixed VM approach will lead to resource waste. Therefore, we propose a runtime elastic VM approach where the cloud can increase or decrease the number of CPUs at different times. This means that when the CPUs are not fully utilized (other resources are busy), we decrease the number of CPUs and when other resources are not fully utilized (CPU busy), we increase the number of CPUs. With runtime elastic VM, the cloud guarantees the number of CPU hours for each user job, yet substantially improve resource utilization of the cloud. Thus, the cloud system can accommodate more jobs.

We illustrate the elastic VM strategy by a toy example in Fig. 1. Assume the cloud provider has a 3-CPU machine. Also assume there are 3 users, each has a MapReduce job. Every MapReduce job has a set of tasks (more specifically, 4 map tasks and 1 reduce task; see Section I.A for a brief background on MapReduce). Assume the processing time of all tasks is 100 seconds. Assume each user rents one VM with two CPUs for a period of 300 seconds to process the MapReduce job. If the cloud uses the conventional fixed VM strategy, the result

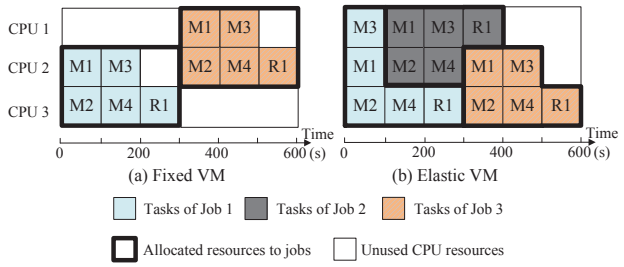


Fig. 1: CPUs Adjustment: (a) Fixed Strategy; (b) Elastic Strategy.

is in Fig. 1(a). The cloud can only accommodate two user jobs and has to reject one. However, if the cloud uses an elastic VM strategy, the cloud can accommodate all 3 jobs, a 50% improvement, while still completing every user job in 300 seconds (see Fig. 1(b)).

Note that the objective of the cloud providers is to accommodate more jobs, not to run an individual job as fast as possible. From the above example we can see the intrinsic reason that an elastic VM strategy accommodates more jobs is that it can better utilize resources as compared to a fixed VM strategy. For example, the utilization of CPU 1, CPU 2, and CPU 3 is 66.7%, 83.3%, and 100% in elastic VM strategy and 33.3%, 83.3%, and 50% in fixed VM strategy. In this paper, we fully explore this idea. We face three challenges: 1) Maintain the existing cloud services; in other words, the elastic VM scheme can be incrementally deployed to existing I-PaaS services. The cloud computing has seen a development for more than a decade, and many services are in use. We believe that it is necessary that we are more aware of incremental deployment as compared to earlier works; 2) A scheme that can maximize the resource optimization; 3) We need implementation and evaluation of the prototype system. The implementation is non-trivial as we target on the cloud provider side; and we need to establish a cloud environment of our own. In particular, we need to ensure the infrastructure-level resource supply and platform-level resource usage consistency: when changing the configuration of a VM, we need to simultaneously modify the parallel task number of big data system installed in the MR-cluster.

The remaining part of the paper is organized as follows. we first (Section III) demonstrate that dynamically adjusting cloud resource in runtime is feasible and the overhead is negligible. And we present a measurement study on the unbalance of the CPU and I/O utilization in runtime for benchmark applications. Second (Section IV), we model the big data jobs/tasks and some key parameters which can indicate resource utilization efficiency. We further validate our modeling by experiments. Such modelings assist the resource adjustment decisions in our system design. Third (Section V), we present the design of prototype system Tetris[1] and cloud resource adjustment algorithms. Finally (Section VI), we show an implementation of Tetris and comprehensively evaluate Tetris on a real private cloud platform using Facebook trace and Wikipedia datasets. The results show that Tetris can accommodate 31.3% more jobs than traditional fixed VM strategy.

[1]We choose "Tetris" as the name of our system, as its basic idea is best illustrated by the game Tetris: http://en.wikipedia.org/wiki/Tetris

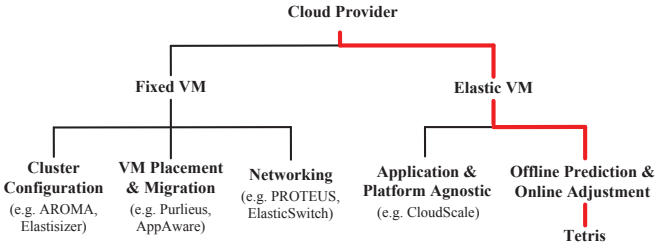


Fig. 2: Design space of Tetris.

II. Related Work

Resource management is always an important issue for the cloud system. As both the cloud system and the application requirements are complex, currently, there is no one clear winning approach that can be applied in every scenario. Related studies look into the resource management problems from different angles. One classification divides the view point by user side and cloud provider side. Our work belongs to cloud provider side research and we categorize this in Fig.2. We first briefly discuss some user side research.

From the view point of user: Users always care about the performance of job processing and the cost for cloud service, so many works study jobs scheduling of the MapReduce systems. In [6][7], scheduling algorithms are proposed for fast completion time. However, these studies did not consider improving cloud resource utilization as their objectives. **From the view point of cloud provider:** given the user requirements, efficiently allocating resources such as CPUs, memories, I/Os, networking, etc, can make the system accommodate more users [8].

I) ***Fixed VM***. Currently, cloud providers allocate fixed VM (Clearly, with full control of the user jobs, the cloud provider can even mix different user jobs on a single VM). Under such context, people study:

1) *Cluster Configuration*. This is also called resource provisioning. To optimize the performance of MapReduce systems and reduce cost paid for renting resources in the cloud, AROMA [9] builds model to provision cloud resources for building the cluster. Herodotou et al. [10] developed Elastisizer to which users can express their cluster sizing problems as queries in a declarative fashion. However, none of the above mentioned studies consider the dynamic cluster with elastic VMs, only focus on the number of computing nodes or instance types.

2) *VM Placement & Migration*. To achieve system load balancing, or switching servers into sleep mode to conserve energy, many literatures study on: a) VM placement before running jobs. For example, Purlieus [11] allocates VMs for MapReduce cluster in a data-locality manner to optimize performance of data access in MapReduce system. b) VM migration across host or even across areas. Shrivastava et al. [12] developed AppAware for incorporating inter-VM dependencies and the underlying network topology into VM migration decisions. However, these studies all allocate a fixed amount of resources to a VM and the resources will not change during the lifetime of the VM.

3) *Networking*. Networking, as a shared resource, also at-

tracts a lot of attention. The common objective is to maximize the number of jobs, and provide certain bandwidth guarantee for each job. For example, ElasticSwitch [13] investigate the trade-off between bandwidth guarantee, fairness and network utilization. In PROTEUS [14], it is observed that the MapReduce applications have different network bandwidth requirements at different stage of the job execution. However, these studies have not considered the requirement characteristics of computing resource of big data applications.

II) ***Elastic VM***. There exists studies on elastic VM. For example, Shen et al. [15] presented CloudScale to employ online resource demand prediction and prediction error handling to achieve adaptive resource allocation. In other words, they are application and platform agnostic. However, without prior analysis of application characteristics, online resource prediction is usually less accurate and less efficient.

In this paper, we present an elastic VM scheme targeting big data applications. Owing to the predictability and periodicity of diverse resource requirement of MapReduce-like jobs, we show that we are able to predict resource offline and adjust resource online. In addition, we specifically try to maintain existing I-PaaS cloud services, so that our work can be incrementally deployable to I-PaaS services like Amazon EMR. On the contrary, past Fixed VM schemes and past Elastic VM schemes targeting on application agnostic resource management avoid such concern. To the best of our knowledge, our work is the first to study an elastic VM strategy based on an I-PaaS cloud system, such as Amazon EMR.

III. The Idling CPUs: A Measurement Study

In this section, we show through measurement the resource unbalance issues for big data jobs. We further investigate the impact of adjusting VCPUs on resource utilization efficiency.

A. Measurement Setup and VCPU Adjustment Implementation

Our cluster is built on a private cloud. In this cloud, there are 18 machines connected with a 1Gbps switch. Every machine has 24 CPU cores and 80GB RAM. We use Xen-4.1.5 to virtualize the physical machine and build our cloud. We allocate 1 VM on every physical machine and build a 18-nodes cluster for our experiments. By default, every VM has one VCPU. In our cluster, we install Hadoop 1.2.1. In our experiments, we configure 1 map slot for every VCPU (default in Hadoop). This means that at most 1 map task can run on one VCPU. The number of reduce task on a VM is set to 1, also the default value of Hadoop.

We run three typical MapReduce algorithms: PageRank, Wordcount and Sort. PageRank is widely used in search engines to calculate the importance of webpages. Wordcount and Sort are commonly used as benchmark applications to evaluate the performance of big data processing [16]. The input data for Wordcount and PageRank is a 75GB document package from Wikipedia [17]. The input data for Sort is a 30GB dataset generated by a random writer.

We conduct two sets of experiments. First, we fix the VCPU number in every VM to one; and we measure the resource utilization unbalance in PageRank, Wordcount and Sort. Second, we dynamically increase the VCPU number and investigate impact of such change on real systems. We use the command set "***xl***" of the Xen-tools to change VCPU number of a VM in runtime. We observe that the overhead for changing VCPU number is negligible.

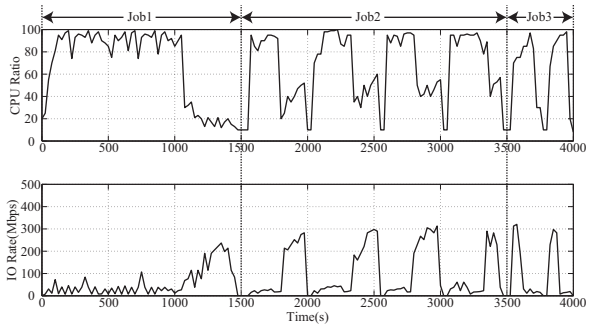


Fig. 3: CPU utilization and I/O rate in PageRank.

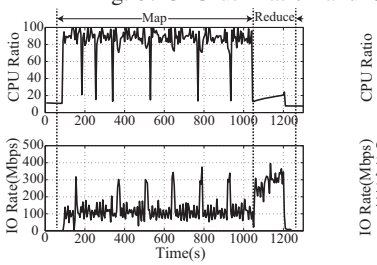


Fig. 4: CPU utilization and I/O rate in Wordcount.

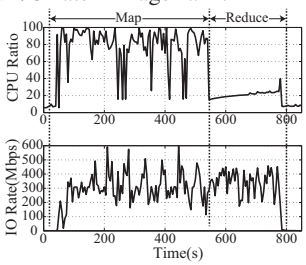


Fig. 5: CPU utilization and I/O rate in Sort.

In this paper, we balance the resources utilization by adjusting the number of VCPUs. In other words, when the VCPU becomes the bottleneck and the utilization of another resource is idle, we increase the VCPU number to increase the utilization of the other resource; and vice versa. We see that increasing/decreasing the VCPU number has an equivalent effect to decreasing/increasing another competing resource. In our measurement, we observe unbalance usage of CPU and I/O. The details are as follows.

B. CPU-I/O Usage Unbalance

1) PageRank: We first show the CPU and I/O utilization in PageRank (see Fig.3). PageRank has three stages: 1) stage 1, extracts webpage connections from the web page data; this job is from 0s to 1500s; 2) stage 2, calculates the rank of every webpage; this job is from 1500s to 3500s; 3) stage 3, sorts ranks of all webpages from 3500s to 4000s.

In map phase of job 1 (from 0s to 1060s), the CPU is fully utilized while average I/O rate is only 18.9Mbps. In more detail, we see that a large amount of map tasks are queued. This is because the number of parallel map tasks is much smaller than the total number of map tasks. In reduce phase (from 1060s to 1500s), average CPU utilization is 18.4% while average I/O rate is 138.6Mbps. Though the two jobs in the remaining two stages have different patterns in CPU and I/O usage, they are also unbalanced in the CPU and I/O usage.

2) Wordcount and Sort: We then conduct a deeper investigation with two benchmark applications: Wordcount and Sort. Each application has only one MapReduce job.

For Wordcount (see Fig. 4), we see that in map phase, the CPU is fully utilized while the average I/O rate is 134.1Mbps. In reduce phase, CPU utilization is 17.2% and I/O rate is 309.3Mbps. These results show that map tasks in Wordcount are CPU intensive and reduce tasks in Wordcount are I/O intensive. For Sort (see Fig. 5), we see that the average CPU utilization in the map phase is 78.3% while the average I/O rate is 325.8Mbps. In reduce phase, the average CPU utilization is 20.8% and the average I/O rate is 350.4Mbps. Thus, both map tasks and reduce tasks in Sort are I/O intensive, but in reduce phase CPU is light-weight.

TABLE I: Average task completion time and average I/O rate of map phase in Wordcount under different VCPU number.

VCPU number	1	2	4	8
Average task completion time (s)	16.7	17.1	21.3	35.5
Completion time of 8 tasks (s)	133.6	68.4	42.6	35.5
I/O rate (Mbps)	134.1	234.4	380.3	407.4

TABLE II: Average task completion time and average I/O rate of map phase in Sort under different VCPU number.

VCPU number	1	2	4	8
Average task completion time (s)	6.4	9.7	17.1	40.5
Completion time of 8 tasks (s)	51.2	38.8	34.2	40.5
I/O rate (Mbps)	325.8	452.7	432.5	341.2

C. The Impact of VCPU Number on Resource Utilization

We next see the impact of adjusting the VCPU number in a VM. The results of Wordcount are shown in Table I. We focus on the map phase on one VM. To understand these results, we randomly choose 8 tasks and consider their executions in the VM. When there is one VCPU, a single task runs 16.7s. Because one VCPU can handle one task, finishing these 8 tasks needs 8 *rounds*. This makes the completion time of 8 tasks to be $16.7 \times 8 = 133.6$s. The I/O rate is 134.1Mbps. When the VCPU number is increased to 2, 2 tasks can be run in-parallel. The completion time of one task increases to 17.1s. This slight increase is due to certain contention among parallel tasks. However, there are only 4 rounds. These 8 tasks finish in $17.1 \times 4 = 68.4$s which is only 51.2% of that of 1 VCPU. The I/O rate increases from 134.1Mbps to 234.4Mbps, an increase of 74.8%. When we increase to 8 VCPUs, we see that completion time of the 8 tasks reduces to 35.5s; only 26.6% of that of the 1 VCPU situation.

We also see that decreasing completion time of the 8 tasks is achieved with increasing costs. When the VCPU number grows, the contention among tasks increases. The average task completion time increases from 16.7s to 35.5s. As a result, the cost of improvement increases. For example, when VCPU number is doubled from 1 to 2, the improvement in completion time of 8 tasks is 48.8% (from 133.6s to 68.4s). However, when we double VCPU number from 4 to 8, the improvement is only 16.7% (from 42.6s to 35.5s).

This is an important observation. Note that our objective is efficient resource utilization of the cloud system to accommodate more jobs, not to complete one MapReduce job as fast as possible. Thus, it is important to be aware of the cost of adding VCPUs and choose a proper VCPU number for a VM. We use *efficiency turning point (ETP)* to indicate the resource utilization efficiency when adding VCPUs to a VM. Intuitively, ETP is the VCPU number where both CPU and I/O resources in a VM are fully utilized. Apparently, ETP depends on type of jobs. In Table.I, we see ETP is 4 for map phase of Wordcount.

We next study Sort, see result in Table II. One important observation is that the I/O rate in Sort reaches its peak when the VCPU number is 2. There is even a decrease when the VCPU number increases from 4 to 8, which means the parallel map tasks suffer severe competition of I/O bandwidth. This is because the map tasks in Sort are both CPU and I/O intensive. When the VCPU number increases, the total I/O bandwidth required by these tasks reached the limit early. Thus, the ETP for the map tasks in Sort is 2, less than that of Wordcount.

We see ETP is an important parameter to indicate resource utilization efficiency and it varies in different scenarios. We will formally define and model ETP in Section IV.

D. Summary

In summary, we see that: 1)There is a common phenomenon that cloud resource usage unbalance between CPU and I/O for MapReduce-like jobs; 2)Increasing the VCPU number will improve job completion time and resource utilization efficiency; yet an excessive increase can come at a cost of the competition on I/O resource, reducing the resource utilization efficiency; 3)To better utilize resources, the best VCPU number of a VM can be represented by ETP which varies in different jobs.

IV. Elastic Job Completion Time Modeling (EJCT)

In this section, we model the completion time of MapReduce tasks/jobs given elastic VMs. We need to define and formulate three important parameters: ETP, wave, and Wave Cutting Point (WCP). Such modeling will assist Tetris resource adjustment decisions. We comment that the modeling is feasible because the MapReduce jobs have fairly standard task sequences. More importantly, big data analysis (e.g., PageRank) usually runs periodically [18]. This means that the same job (same data size, same function to process the data) will run again and again. We can thus construct/train model parameters in the first period.

A. The Task Completion Time Model (TCTM) and ETP

The completion time of a task is determined by the "supply" and "demand". The supply is determined by the resources in a VM. The demand is determined by the resource requirement of a single task and the number of parallel tasks in the VM. In what follows, we first model the supply side and the demand side. Then we present the overall model.

1) The Demand Side: The amount of resources required by a single task is determined by two factors: 1) the function (application) of the task; and 2) the size of data processed by the task. Intuitively, function means what the work is and decides how many resources needed to process one unit of data. Input data size decides how much work to do.

We first model the function of the task. It has been observed that MapReduce jobs have *resource signatures* [11][14]. A

resource signature means the required I/O bandwidth, memory and network are relatively fixed given a specified CPU. For example, in map phase of Wordcount (see Fig.4), when CPU is fully utilized, I/O rate stays around 134.1Mbps. Thus, we use resource signatures to represent resource demands of a function. Formally, let c_i be the function executed by task i. Let T_{c_i} be the time for c_i to process one unit of data. Let S_i be the set of resources that task i requires. Let $o_{c_i}^{(k)}$ be requirement of resource type k for c_i.

We next model the size of data processed by the task. For a MapReduce job, the input data size is same for the tasks in a phase. In the map phase, the input data are split into blocks of same size. Before the reduce phase, a MapReduce system tends to evenly distribute the intermediate data to every reduce task. In this paper, we assume an even distribution. Formally, let D_i be input data size of task i.

In the default setting of a MapReduce system, the number of parallel tasks on every VM is the same and this number is set to be proportional to the VCPU number, e.g., one VCPU is associated with one map task and one reduce task. As we change the VCPU numbers, we let the number of parallel tasks proportional to the VCPU number. Formally, let p be the ratio of the number of parallel tasks to the number of VCPUs. p is a constant.

2) The Supply Side: The resources required by the tasks in a MapReduce job are CPU, memory, I/O bandwidth and networking bandwidth. Formally, let n be the VM where task i is executed. Let U_n be the number of VCPUs in VM n. Let $O_n^{(k)}$ be the capacity of resource type k in VM n.

3) The overall model (TCTM) and ETP: The completion time of task i on VM n is:

$$TCT(D_i, c_i, O_n, U_n) = D_i T_{c_i} \prod_{k \in S_i} f_k(c_i, O_n^{(k)}, U_n) \qquad (1)$$

where $f_k(c_i, O_n, U_n)$ models the increase of task completion time caused by the mismatch of resource k supply and demand. Intuitively, when the number of VCPUs increases, pU_n increases and the resource k shared by a single task decreases. There are two scenarios: 1) when the resource supply is greater than the combined demands, there is no increase in task completion time; and 2) otherwise, the task completion time increases proportionally with demand-supply mismatch. We formally formulate these two scenarios as:

$$f_k(c_i, O_n^{(k)}, U_n) = \begin{cases} 1 & o_{c_i}^{(k)} pU_n \le O_n^{(k)} \\ \frac{o_{c_i}^{(k)} pU_n}{O_n^{(k)}} & o_{c_i}^{(k)} pU_n > O_n^{(k)} \end{cases} \qquad (2)$$

These two scenarios in Eq.2 also indicate an ETP.

Definition 1: The Efficient Turning Point (ETP) of a resource k for function c_i is the maximum VCPU number where $o_{c_i}^{(k)} pU_n \le O_n^{(k)}$. Thus, $ETP(c_i, O_n^{(k)}) = \lfloor \frac{O_n^{(k)}}{o_{c_i}^{(k)} p} \rfloor$.

This model means that when the supply of resource k cannot satisfy the total demand of resource k, the resource k will become a bottleneck and adding more VCPUs will lead to a waste of the VCPU resources.

B. The Job Completion Time Model (JCTM), waves, and WCP

1) Waves: The tasks in a MapReduce job run in parallel, i.e., a subset of tasks start at roughly the same time, process the same data size, and finish at similar times. Therefore, we can consider the tasks run in *waves*[19].

2) JCTM: We first model the completion time of each wave of map tasks and reduce tasks. The job completion time is then a summation of all the waves.

We define a *resource allocation plan* of a MapReduce job as a sequence of CPU allocations for every VM in a cluster. Let N be the number of VMs in a cluster for a MapReduce job. Let L and K be the number of map and reduce tasks in this job. Let $W^{(M)}$, $W^{(R)}$ be the number of waves in the map and reduce phases. The resource plan of the map and reduce phases $A^{(M)}$, $A^{(R)}$ are arrays of $N \times W^{(M)}$ and $N \times W^{(R)}$.

An element $U_{nm}^{(M)}$ in $A^{(M)}$ represents the VCPU number on VM n in the m-th wave in the map phase. An element $U_{nm}^{(R)}$ in $A^{(R)}$ represents the VCPU number on VM n in the m-th wave in the reduce phase. Let $c^{(M)}$, $c^{(R)}$ be the functions of the map and reduce phases. Let $D^{(M)}$, $D^{(R)}$ be the input data sizes of a map and reduce task. The completion time of a MapReduce job with resource plan $A^{(M)}$, $A^{(R)}$ is:

$$\begin{aligned} JCT(A^{(M)}, A^{(R)}) = & \sum_{m \in [1, W^{(M)}]} TCT(D^{(M)}, c^{(M)}, O_n, U_{nm}^{(M)}) \\ & + \sum_{r \in [1, W^{(R)}]} TCT(D^{(R)}, c^{(R)}, O_n, U_{nr}^{(R)}) \end{aligned} \qquad (3)$$

3) Wave Cutting Point (WCP): From Eq. 3, we see that the completion time of a MapReduce job (JCT) is determined by two factors: 1) the completion times of the tasks, and 2) the number of waves. Intuitively, there can be a trade-off between the tasks and waves. If more tasks run in-parallel, the tasks can finish in fewer number of waves. We define *wave cutting point (WCP)* to indicate this.

Definition 2: Let V be the number of tasks in a phase. A Wave Cutting Point (WCP) is a number of total VCPUs in a cluster satisfying $\lceil \frac{V}{WCP} \rceil < \lceil \frac{V}{WCP-1} \rceil$.

More specifically, for a WCP, when the VCPU number increases from WCP-1 to WCP, the number of waves in the job decreases. For example, suppose a job has 20 map tasks and every VCPU processes one task at a specified time. If the total number of VCPU in the cluster is 4, the job finishes in 5 waves. When the VCPU number increases from 4 to 5, the number of waves decreases from 5 to 4. Thus 5 is a WCP. Similarly, 10, 20 are also WCPs.

C. Validation

1) Validation Setup: We use same setup used in measurement. For simplicity we mainly analyze Wordcount benchmark in this section. To validate our models, we initialize parameters in the TCTM: D_i, T_{c_i}, U_n, O_n, $o_{c_i}^{(k)}$ and p by measurement. We consider three resources: I/O bandwidth, memory and network bandwidth. Every VM has a fixed amount of I/O bandwidth, memory and network bandwidth. In this paper, we focus on the number of VCPUs. Thus, we concentrate on

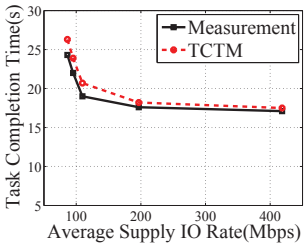


Fig. 6: TCTM Validation.

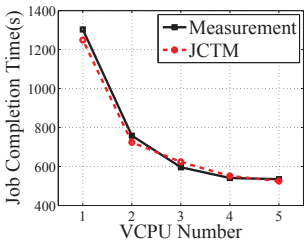


Fig. 7: JCTM Validation.

validating the prediction accuracy of our model when VCPU number changes. First, we set the number of VCPU to 1 on every slave. Then we run benchmark applications on the cluster and measure the task and job completion times. Second, we change the number of VCPU of every VM from 1 to 5. The number of parallel tasks on every VM is adjusted accordingly. In every iteration, we run the same jobs on the cluster.

2) Validation Results: **Task Completion Time Model.** When VCPU number increases, parallel task number grows accordingly. Total I/O bandwidth is fixed for all VMs. As a result, I/O bandwidth shared by every task decreases. We compare measured task completion time with results predicted through our model when I/O bandwidth shared by every map task changes. Fig.6 shows that when the supply I/O bandwidth per task is higher than I/O demand, average completion time of map tasks is stable. However, when supply I/O rate is lower than the demand, task completion time increases rapidly. More specifically, when supply I/O rate range from 196.8Mbps to 418.4Mbps, which is larger than the demand (133.6Mbps), task completion time is about 17.5s both in experiment and model prediction. In measurement, when shared I/O bandwidth is 109.6Mbps, less than the demand, task completion time grows to 19.2s. After that, task completion time rockets to 24.3s when shared I/O bandwidth is 86.4Mbps. We see that the deviation between experiment result and model prediction is small, which is always less than 9.5%. These results indicate that our model can predict task completion times accurately.

Job Completion Time Model. Next, we further validate the job completion time model (Eq.3). The result is shown in Fig.7. We see that when VCPU number is increased from 1 to 5, job completion time in experiment decreases from 1303s to 535s while model-predicted job completion time decreases from 1249s to 524s. The deviation between experiment result and model-predicted result is small. The largest deviation is 64s. We can see that two results follow the same trend and the deviation between them decreases to 2.1% when VCPU number increases from 1 to 5.

V. Tetris Design and Adjustment Algorithms

We now present the design of Tetris system. We first introduce the overall architecture of Tetris. We then present the elastic VM allocation algorithms (eVM). In Section VI, we present the details for a Tetris prototype implementation.

A. Tetris Architecture

Tetris works for an I-PaaS cloud supporting big data applications, e.g., Amazon EMR. Such cloud has two components: 1) a cloud administrator, which admits user jobs, creates user clusters and maintains the big data system on top of the user clusters; and 2) big data systems, e.g., Hadoop, which are constructed on top of the clusters. The big data system monitors the VMs/nodes in the cluster and schedules the tasks of the user job. The flow of such cloud works as follows. When a user job arrives, the user specifies a cluster configuration, i.e., the requested resources. The I-PaaS cloud determines whether to admit this job or not. The decision is usually based on the available resource in the cloud, and other concerns such as security, etc. If the job is admitted, the cloud administrator allocates a cluster in the data center, constructs a big data system on top of the cluster and then the job is executed.

Tetris slightly modifies the flow of the I-PaaS cloud as follows. When a user job arrives, the user specifies a cluster configuration, i.e., the requested resources. The I-PaaS cloud determines whether to admit this job or not. If the job is admitted, the cloud administrator allocates a cluster, constructs the big data system on top of the cluster. Then Tetris is triggered to determine a *resource allocation plan*. The resource allocation plan specifies the resources needed at each individual time of the user job life cycle, which can be different. Then the job is executed. During runtime, the resources are adjusted according to the resource allocation plan.

The architecture of Tetris is shown in Fig.8. Tetris has two computing modules which computes the resource allocation plan; and two interface drivers which are triggered during the job execution to realize the VCPU adjustment specified in the resource allocation plan.

The two interface drivers are: 1) an Elastic VM-Cloud (EVMC) driver; it offers the interfaces for Tetris to change the number of VCPUs of a VM in the cloud. It also reports the resource status information, such as the used and idle CPU numbers on every physical machine, to Tetris; and 2) a user cluster (UC) driver; it offers the interfaces for Tetris to notify the big data system scheduler to change the number of parallel tasks when the VCPU number changes. To assist Tetris decision making, UC driver reports the job profiling data, such as job function and job input data size, to Tetris. The resource utilization information in a VM (e.g. CPU utilization and I/O bandwidth utilization) is reported to Tetris, too.

The two computing modules are: 1) an EJCT module; EJCT computes TCTs, JCTs, ETPs, WCPs etc of the user job; and 2) an Elastic Resource Allocation (ERA) module; after a job is admitted and its cluster is allocated, ERA computes the resource allocation plan.

In our Tetris/ERA design, ERA computes the resource allocation plan using JCT, ETP, etc that EJCT computed. This can be seen as a proactive design. In our experiment, we will compare Tetris and a reactive design. In the reactive design, we remove the EJCT module from Tetris. ERA makes resource adjustment based on monitoring the resource utilization. We see that our proactive Tetris design has better performance.

We want to comment Tetris from the user's point of view. Without Tetris, a user rents a fixed cluster with certain number of VCPUs and time period (CPU-Time). He then submits the

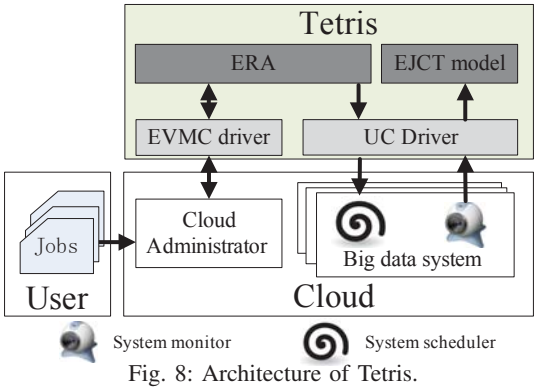


Fig. 8: Architecture of Tetris.

job. Amazon EMR usually notifies the user for renewal before the rental time expires. When the job completes, Amazon will charge the user according to the infrastructure consumed, plus around 25% of additional cost for the Hadoop system. Tetris guarantees that both the total CPU-Time and user job completion time are no worse than the fixed VM strategy. Thus, the cloud provider can do the same as usual.

B. Tetris Elastic VM Adjustment Algorithms (eVM)

When a job is admitted, ERA module call eVM algorithm to compute the resource allocation plan for this job. ***The objective*** is to accommodate as many jobs as possible with the constraints of guaranteeing the job completion time and the total CPU-Time the user registered for the cluster.

To maximally accommodate jobs, we try to execute each job efficiently. For each incoming job, we study two trade-offs.

First, there is unbalance usage for different types of resources. When unbalance occurs, at least one type of resource is wasted. Recall that the ETP for a type of resource is the VCPU number that can make this type of resource fully utilized. Though there are ETPs for I/O, memory, network bandwidth, we only need to consider the first type of resource that introduces resource utilization unbalance. We develop our first algorithm eVM-base() which sets the VCPU number to 1) the ETP, the closer the better; and 2) the limit of total CPU number of the physical machines. We do not increase beyond the limit of the physical machines, as this will need to migrate memory etc.

Second, recall that the tasks of a MapReduce job run in waves. Setting the VCPU number according to the ETP only may introduce additional waves for very few tasks. We found that these "lingering" tasks, together with the limit of physical available CPU number of the machines may reject jobs that will be accommodated otherwise. To this end, we develop our Elastic VM Adjustment algorithm eVM() where we slightly compromise ETP according to WCPs. Recall that WCPs are the VCPU numbers that cut the waves. We increase ETP to 1) the closest WCP; and 2) JCT (the job completion time) decreases.

To guarantee the CPU-Time, in both eVM-base() and eVM(), we compute in each time slot through EJCT the total VCPU-Time needed to complete the remaining tasks. If this is less than the user VCPU-Time, we still can increase/decrease the VCPU number; otherwise, we complete the remaining tasks using Fixed-VM. Thus, the user job will not exceed the VCPU-Time that the user ordered.

Algorithm 1 eVM-base() with ETP

Input: $\langle \mathbb{H}, \mathbb{C}^{ini} \rangle$, $\mathbb{R}^{ini}$, ETP, L, CT^{user}.
Output: $\langle \mathbb{H}, \mathbb{C} \rangle$, $\mathbb{R}$.

$\mathbb{C} = \mathbb{C}^{ini}$; $\mathbb{R} = \mathbb{R}^{ini}$;
Compute current job completion time JCT^{cur};
for $\forall t \in [0 \dots JCT^{cur}]$ **do**
  for $\forall n \in [1 \dots N]$ **do**
    // ETP constraint and JCT guarantee.
    if $c_n < ETP$ **then**
      // physical CPU number constraint.
      $c = \min(ETP - c_n, R_{h_n})$;
      Compute CT^{ela} after adding c VCPUs to VM_n;
      // user VCPU time constraint.
      if $CT^{ela} < CT^{user}$ **then**
        $c_n = c_n + c$; $R_{h_n} = R_{h_n} - c$;
  Compute JCT^{new} after adjusting VCPUs;
  $JCT^{cur} = JCT^{new}$; $c^{total} = \sum_{n \in [1 \dots N]} c_n$;
  // user remaining VCPU time.
  $CT^{user} = CT^{user} - c^{total} \times interval$;
return;

Algorithm 2 eVM() with ETP and WCP

Input: $\langle \mathbb{H}, \mathbb{C}^{ini} \rangle$, $\mathbb{R}^{ini}$, ETP, L, CT^{user}.
Output: $\langle \mathbb{H}, \mathbb{C} \rangle$, $\mathbb{R}$.

$(\mathbb{C}, \mathbb{R})$ = eVM-base$(\mathbb{H}, \mathbb{C}^{ini}, \mathbb{R}^{ini}, ETP, L, CT^{user})$;
Compute current job completion time JCT^{cur};
for $\forall t \in [0 \dots JCT^{cur}]$ **do**
  $c^{total} = \sum_{n \in [1 \dots N]} c_n$;
  for $\forall n \in [1 \dots N]$ **do**
    // no idle CPUs.
    if $R_{h_n} == 0$ **then continue;**
    // ETP constraint
    for $\forall i \in [1 \dots ETP]$ **do**
      // physical CPU number constraint.
      if $i > R_{h_n}$ **then break;**
      // cut waves.
      if $\lceil L/c^{total} \rceil > \lceil L/(c^{total} + \min(R_{h_n}, i)) \rceil$ **then**
        Compute JCT^{new} after adding i VCPUs to VM_n;
        // JCT guarantee.
        if $JCT^{cur} > JCT^{new}$ **then**
          Compute CT^{ela} after adding i VCPUs to VM_n;
          // user VCPU time constraint.
          if $CT^{ela} < CT^{user}$ **then**
            $c_n = c_n + i$; $R_{h_n} = R_{h_n} - i$;
            $JCT^{cur} = JCT^{new}$; $c^{total} = c^{total} + i$;
            break;
  // user remaining VCPU time.
  $CT^{user} = CT^{user} - c^{total} \times interval$;
return;

C. Algorithm Details

We now present the algorithm details. Both eVM-base and eVM are greedy-based. They all output a resource allocation plan, i.e., the VCPU number in each time slot (here we use *interval* to denote the time slot).

We first present eVM-base. It computes the VCPU number iteratively for each time slot. There are 3 constraints to determine the VCPU number (or whether we should increase or decrease the VCPU number as compared to the Fixed-VM strategy in that time slot): 1) the ETP, the closer the better; 2) the limit of the physical CPU number. Note that we do

not increase beyond the physical limit as this will need to migrate memory etc; and 3) in each iteration, we compute through EJCT the total VCPU time needed to complete the remaining tasks. If this is less than the user VCPU time, we still can increase/decrease the VCPU number; otherwise, we complete the remaining tasks using Fixed-VM. The philosophy behinds this is that we have to complete the user job without exceeding the VCPU time that the user purchases. In summary, we increase/decrease the VCPU number to the closest of ETP where constraints 2) and 3) are not violated.

We can see that our algorithm will not violate the VCPU time. In each iteration, we can only improve the ETP. The job completion time can only improve and thus also guaranteed. eVM is developed similarly where we have an additional adjustment for ETP to the closest WCP where JCT decreases. Formally, the pseudo codes of eVM-base and eVM are showed in Algorithm 1&2.

Notations: Assume the VMs of the job is placed on a cluster contains N physical machines and let $\mathbb{H} = \{h_1, h_2, \ldots, h_N\}$ be the id array of the physical machines. Let $\mathbb{C} = \{c_1, c_2, \ldots, c_N\}$ be VCPU numbers allocated to the VMs. Let $\mathbb{R} = \{R_1, R_2, \ldots, R_N\}$ be the available CPU numbers in each physical machine, i.e., that is possible for the elastic strategy to increase the VCPU number. Let $\mathbb{C}^{ini}$ be the initial allocated VCPU array for the VMs that scheduled to process the job. And let $\mathbb{R}^{ini}$ be the initial remaining VCPUs array for each physical machines. For an arrival job, we can compute the ETP of the code, the number of tasks L and the parameters used in Eq.3 for computing JCTM. Let CT^{user} be the user VCPU time and CT^{ela} be the total VCPU time that needed to complete the remaining tasks.

VI. Implementation and Experiment

A. Implementation

We develop a prototype of Tetris. Tetris is used for an I-PaaS cloud provider. Tetris needs to change the configuration of a VM and modify the big data system installed in the cluster. Current public available cloud such as Amazon EMR, Microsoft HDInsight do not offer such authorities, yet. Therefore, before we develop Tetris, we first implement a prototype cloud system similar to Amazon EMR. Our cloud system is developed based on Xen, a common virtualization platform used by Amazon, Rackspace, etc. For the big data system on top of the user clusters, we use Hadoop 1.2.1, which is the same version adopted by Amazon EMR. Hadoop has two types of nodes: master and slave. One VM of the cluster is configured as the master node and the remaining VMs in the cluster are configured as the slave nodes. The master node monitors slave nodes and controls the tasks running on slave nodes. Slave nodes process tasks.

We then implement Tetris. The implementation framework is shown in Fig. 9. We develop the EVMC and UC drivers for Hadoop and Xen. For the EVMC driver, we register a handle to the cloud administrator so that Tetris is informed when the resource utilization changes, e.g., when a cluster is released. When Tetris changes the VCPU number of a VM, this change

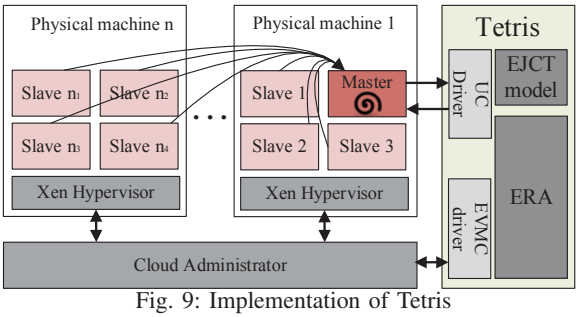


Fig. 9: Implementation of Tetris

is realized by invoking the Xen APIs on the Xen hypervisor where the VM allocated. For the UC driver, we monitor and control Hadoop through the master node. We install a monitor script on every slave node. Thus a master node can collect the resource utilization on every slave node. The UC driver fetches these information from the master node directly. Note that we need to inform Hadoop to increase/decrease in-parallel tasks running on a VM when the VCPU number of a VM changes. The default scheduler of Hadoop, however, assumes that all slave nodes have the same CPU resources. We thus develop a scheduler for the Hadoop system. In our scheduler, we add a list to record the current VCPU numbers on every slave node. Different scheduling logic can be used.

B. Experiment Setup

Cloud environment. In the experiment, we use same cloud environment setting as in the measurement. Different from the measurement where we run only one cluster in the cloud for detail investigation, Tetris runs multiple jobs simultaneously. Thus, we run multiple clusters in our evaluation.

Workloads. We study two types of jobs, Wordcount and Sort. The workload injected for our system is determined by the job size and the real data. The job size is based on Facebook trace [20] in which many jobs are small (less than 2 map tasks) yet there are big jobs with more than 500 map tasks. The real input data of Wordcount are from Wikipedia and the input data of Sort are generated by a random writer.

To have a fair comparison, we develop a job set generator to create the same sequence of jobs for every algorithm. We generated 150 jobs from Facebook trace. The distribution of the job inter-arrival time was exponential with a mean of 10 s. The experiment period is 1500s.

Comparison strategy. We compare performance of three strategies: eVM, eVM-base and Fixed-VM (current strategy of Amazon EMR). In Fixed-VM, the configurations of VMs do not change once they are initialized.

C. Experiment Results

1) An overview: **Throughput**. Fig.10 show the throughputs of three algorithms. X-axis is the time after the submission of the first job of the workload. Y-axis is number of admitted jobs. We can see that eVM-base and eVM algorithm have greater throughputs than Fixed-VM. Fixed-VM only admits 53.3% jobs at the time of 1500s. eVM-base and eVM admit 64% and 70% jobs respectively. Compared to Fixed-VM, eVM-base has an 18.8% improvement, and eVM has a 31.3% improvement.

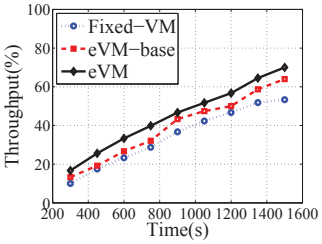


Fig. 10: Under load balancing.

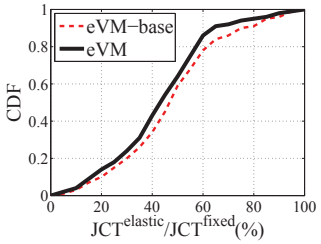


Fig. 11: JCT guarantee.

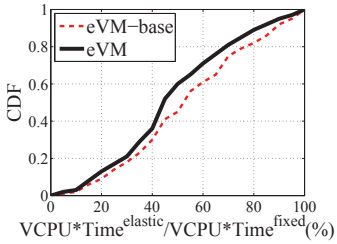


Fig. 12: VCPU time constraint.

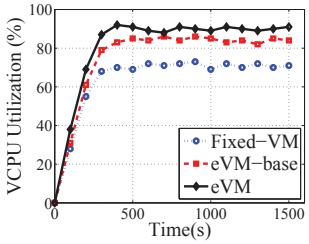


Fig. 13: VCPU Utilization.

Comparing the results between WordCount and Sort, we observe that the improvement in Sort is greater than that of Wordcount. In our measurement section, however, we see Sort has greater I/O than Wordcount and the ETP of the map tasks in Sort is less than the ETP of the map tasks in Wordcount. In other words, when increasing the VCPUs, the Sort jobs gain less benefit than the Wordcount jobs. After a detailed investigation, we found the high improvement of Sort comes from reclaiming VCPUs in the reduce phase. The reduce phase in Sort occupies 37% of whole job processing time while the reduce phase in Wordcount only occupies 21%. This means that if a Wordcount job and a Sort job have the same JCT, eVM can release VCPUs earlier in Sort. Thus, more jobs can be admitted in the Sort workload. This observation reveals that the improvement of eVM comes from not only increasing VCPUs when there are idle VCPUs outside the cluster but also from releasing VCPUs in the cluster when workload in the cluster is low.

Job completion time guarantee. We then study the job completion time. From user's point of view, the user job completion time under elastic VM strategy should not be greater than the completion time under the default Fixed-VM strategy. For every job executed in eVM we compute the ratios of JCT in eVM divided by JCT in Fixed-VM. Note that this ratio should always less than 100%. Fig.11 shows the CDF of the ratios. We see that in both eVM and eVM-base, every job has a JCT less than the JCT in Fixed-VM. This means the completion time of all jobs are guaranteed. Moreover, we see 86% jobs in eVM finish with 60% of JCT in Fixed-VM.

User VCPU time constraint. In an I-PaaS cloud, the users rent VCPU time from the cloud providers. Tetris should guarantee that the VCPU time under elastic strategies is no greater than the Fixed-VM strategy, so as to avoid over charging the users. Fig.12 shows that eVM and eVM-base can guarantee all the jobs can be finished in VCPU times.

In summary, Tetris not only is good in the total number of jobs accepted (throughput), but also guarantees the job completion time and VCPU time. The reason that Tetris excels in all fronts is because the resource utilization is better. And we now look into the VCPU utilization.

VCPU utilization. VCPU utilization is the percentage of allocated VCPU number in total VCPU number. Fig.13 show the VCPU utilization during job processing. We see that after 500s, the VCPU utilization of eVM is over 91%, eVM-base is around 87%, and Fixed-VM is around 75%. As compared to Fixed-VM, eVM and eVM-base increase the VCPU utilization by 16% and 12% on average.

Tetris in micro view. We now look one step further into Tetris VCPU utilization. We look at Tetris in micro view. To facilitate this, we conduct a finer-controlled experiment. We construct a three-job workload as follows: job 1 has 400 map tasks and 2 reduce tasks; job 2 has 300 map tasks and 2 reduce tasks; and job 3 has 100 map tasks and no reduce tasks. Job 1 is submitted at time 0; job 2 is submitted after the reduce phase of job 1 starts, and job 3 is submitted after job 1 finishes.

Fig.15 shows a timeline of the experiment for both Tetris and Fixed-VM. Using Tetris, all three jobs are admitted and finished and using Fixed-VM, job 2 is rejected. Our micro view shows clearly why and how this happens. Using Tetris, the VCPU number of job 1 increased in the map phase (see a comparison of Fixed-VM) and fully utilize all VCPU resources (close to 100%). At time 705s, the map tasks of job 1 finishes and the VCPU utilization of job 1 decreases. Tetris dynamically releases 80% of the VCPU resources to the cloud. At the same time, job 2 is submitted. Job 2 is accepted and it uses the released VCPU resources of job 1. Job 1 finishes at 918s. From 918s to 1056s, job 2 occupies all VCPU resources. Map phase of job 2 finishes at 1067s. And 70% VCPU resources are reclaimed. At 1070s, job 3 is submitted. Job 3 uses the 70% VCPU resources which released by job 2. Note that although we fine-controlled the job submissions in this experiment for illustration, all elastic VM adjustments are done by Tetris.

As a comparison, for the Fixed-VM strategy, even that the VCPU is not fully utilized, the cloud cannot adapt to this. No matter how we control job submissions, the Fixed-VM strategy always fails to squeeze all 3 jobs in the cloud.

2) The Impact of VM placement: Recall that when a job is admitted, the cloud provider will construct the cluster for this job. Tetris is then executed. Clearly, the performance of Tetris depends on the VM placement strategy. We study two typical VM placement strategies: load balancing and server consolidation. In a load balancing strategy, the objective of the cloud is to avoid resources competition. Thus, the cloud places new VMs in a physical machine that has the least load. In the server consolidation strategy, the objective of the cloud is to spare machines, e.g., for power conservation, etc. Thus, the cloud places new VMs in the physical machine which has high load. Fig.10 is under load balancing strategy while Fig.14 is under server consolidation strategy.

Comparing Fig.10 with Fig.14, we see that eVM admits 31.3% more jobs than Fixed-VM under load balancing strategy and admits 20% more jobs under server consolidation strategy. Our elastic algorithms perform better under the load balancing strategy. The reason is that under the server consolidation strategy, VMs are crowded in physical machines, leaving less adjustment opportunities for Tetris. However, we still see that eVM and eVM-base outperform Fixed-VM in such a

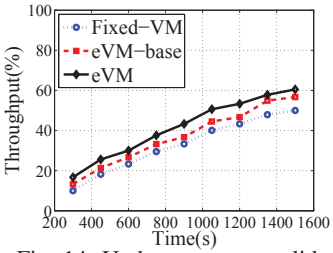
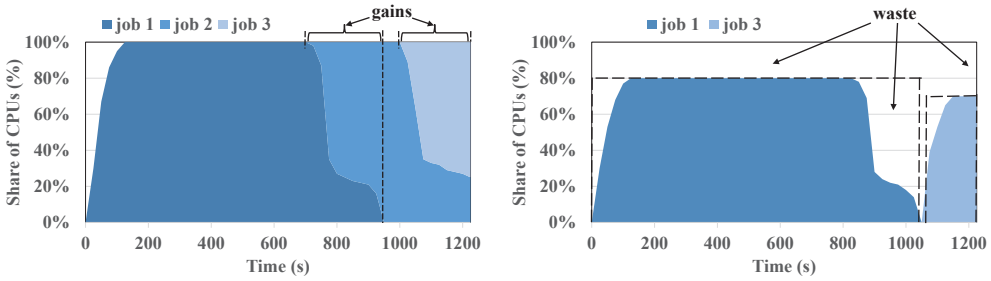


Fig. 14: Under server consolidation.



(a) Tetris (b) Fixed-VM

Fig. 15: Stacked chart showing the percent of occupation of VCPUs in one server in the cluster given to each job as a function of time between Tetris and Fixed-VM.

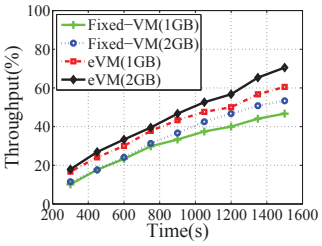


Fig. 16: Impact of memory.

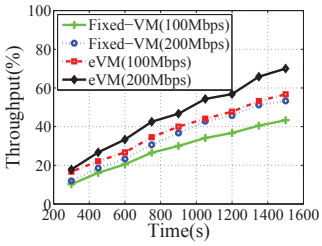


Fig. 17: Impact of bandwidth.

constrained situation.

3) The Impact of Memory and Network Bandwidth: We now decrease memory and network bandwidth from their default value to evaluate the performance of Tetris.

Impact of Memory. We set the memory of each node as 2GB (default) and 1GB (starting to become a bottleneck). Fig.16 shows that when the memory decreases from 2GB to 1GB, both eVM and Fixed-VM suffer from performance degradation. eVM admits 14.3% less jobs while Fixed-VM admits 12.5% less jobs. However, eVM outperforms Fixed-VM consistently.

Impact of Network bandwidth. We set the bandwidth of each node as 100Mbps and 200Mbps. Fig.17 shows that when the bandwidth decreases from 200Mbps to 100Mbps, eVM admits 23.5% less jobs while Fixed-VM admits 18.8% less jobs. Yet, eVM outperforms Fixed-VM consistently.

VII. Conclusion

In this paper, by measurement we observed that there is a common phenomenon that cloud resource usage unbalance between CPU and I/O for MapReduce-like jobs running in an I-PaaS platform such as Amazon EMR. Then we proposed and demonstrated an elastic VM strategy to optimize the resource utilization unbalance problem effectively. Note that when the unbalanced utilization of different types of resources occurs, at least one type of resource is wasted/idle. To best explore this idea, we presented a design and implementation of Tetris with incremental deployment. We showed by comprehensive experiments that Tetris can better utilize resources and consequently accommodate more jobs.

Acknowledgement

This work is supported by the National Basic Research Program of China under Grant No. 2012CB315806, the National High-Tech Research and Development Plan of China under Grant No. 2015AA015601, the National Natural Science Foundation of China under Grant No. 61432009, Specialized Research Fund for the Doctoral Program of Higher Education under Grant No. 20130002110058. Dan Wang is supported in part by National Natural Science Foundation of China No. 61272464, RGC/GRF PolyU 5264/13E, HK PolyU 1-ZVC2, G-UB72, and G-YBAG.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proc. of USENIX OSDI*, 2004.
[2] "Google app engine for mapreduce," 2015. https://cloud.google.com/appengine/docs/python/dataprocessing/.
[3] "Amazon emr," 2015. https://aws.amazon.com/elasticmapreduce/.
[4] "Microsoft hdinsight," 2015. http://azure.microsoft.com/en-us/services/hdinsight/.
[5] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. of USENIX NSDI*, 2012.
[6] F. Chen, M. Kodialam, and T. Lakshman, "Joint scheduling of processing and shuffle phases in mapreduce systems," in *Proc. of IEEE INFOCOM*, 2012.
[7] X. Ling, Y. Yuan, D. Wang, J. Liu, and J. Yang, "Joint scheduling of mapreduce jobs with servers: Performance bounds and experiments," *Journal of Parallel and Distributed Computing*, vol. 90-91, pp. 52–66, 2016.
[8] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: Towards a cloud definition," in *Proc. of ACM SIGCOMM*, 2008.
[9] P. Lama and X. Zhou, "AROMA: automated resource allocation and configuration of mapreduce environment in the cloud," in *Proc. of USENIX ICAC*, 2012.
[10] H. Herodotou, F. Dong, and S. Babu, "No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics," in *Proc. of ACM SoCC*, 2011.
[11] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: Locality-aware resource allocation for mapreduce in a cloud," in *Proc. of ACM SC*, 2011.
[12] V. Shrivastava, P. Zerfos, K.-W. Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee, "Application-aware virtual machine migration in data centers," in *Proc. of IEEE INFOCOM*, 2011.
[13] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "Elasticswitch: Practical work-conserving bandwidth guarantees for cloud computing," in *Proc. of ACM SIGCOMM*, 2013.
[14] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: incorporating time-varying network reservations in data centers," in *Proc. of ACM SIGCOMM*, 2012.
[15] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: Elastic resource scaling for multi-tenant cloud systems," in *Proc. of ACM SoCC*, 2011.
[16] P. Costa, A. Donnelly, A. Rowstron, and G. O'Shea, "Camdoop: exploiting in-network aggregation for big data applications," in *Proc. of USENIX NSDI*, 2012.
[17] "Wikipedia," 2015. http://en.wikipedia.org/.
[18] 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 *Proc. of ACM SoCC*, 2012.
[19] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proc. of USENIX OSDI*, 2008.
[20] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. of ACM EuroSys*, 2010.