Time-Constrained Live *VM* Migration in Share-Nothing *IaaS*-Clouds

Konstantinos Tsakalozos [#1,‡], Vasilis Verroios [*], Mema Roussopoulos [#2], and Alex Delis [#3]

[#] *University of Athens, Athens, 15748, Greece*
{*k.tsakalozos*[1], *mema*[2], *ad*[3]}*@di.uoa.gr*

[*] *Stanford University, Stanford, CA 94305-9040*
verroios@stanford.edu

[‡] *Microsoft, London, EC1N 2ST, United Kingdom*

***Abstract*—Both economic reasons and interoperation requirements necessitate the deployment of *IaaS*-clouds based on a share-nothing architecture. Here, live *VM* migration becomes a major impediment to achieving cloud-wide load balancing via selective and timely *VM*-migrations. Our approach is based on copying virtual disk images and keeping them synchronized during the *VM* migration operation. In this way, we ameliorate the limitations set by shared storage cloud designs as we place no constraints on the cloud's scalability and load-balancing capabilities. We propose a special-purpose file system, termed *MigrateFS*, that performs virtual disk replication within specified time-constraints while avoiding internal network congestion. Management of resource consumption during *VM* migration is supervised by a low-overhead and scalable distributed network of brokers. We show that our approach can reduce up to 24% the stress of already saturated physical network links during load balancing operations.**

I. Introduction

The need to balance load across computing systems that make up an *IaaS* cloud infrastructure is of paramount importance for the quality of the rendered services [1], [2]. In such *IaaS*-clouds that predominantly offer virtual machines (*VM*s), load-sharing can be achieved through *VM* migration; here, a *VM* moves from one physical machine (*PM*) to another. *VM* movement helps offload congested physical nodes, may considerably enhance the utilization of the underlying computer systems, and may ultimately improve the quality of provided services.

In large-scale *share-nothing* infrastructures, *VM* migration is not a trivial task as virtual disks in the order of multiple *GBytes* have to be transferred from source to target *PM*s. Often, *VM* migrations cause significant performance degradation or even yield discernible downtime for applications [3], [4]. Moreover, multiple simultaneous *VM* migrations coinciding with high-load periods in the underlying *VM*s and/or the involved network resources further exacerbate the problem. Relying on usage statistics and taking into account the infrastructure's architecture, an *IaaS*-cloud provider can decide upon a time window for scheduling each pending *VM* migration. A *VM* migration failing to complete within its pre-specified window, can degrade the *QoS* experienced by the affected *VM*s and may lead to a number of Service Level Agreement (*SLA*) violations.

In this paper, we focus on the problem of real–time scheduling of *live VM* migration tasks in *share-nothing IaaS*-clouds: for each *VM* migration task a new *PM* host and a time window for the migration to take place are pre-defined by the cloud reallocation policy. Moreover, migrations are *live* since they are performed while the migrating *VM*s remain on–line and involve a short downtime hardly noticeable by users interacting with the *VM*s [5], [6]. In this context, a real–time scheduling mechanism must:

- control the amount of resources allocated on each migration task, based on the *QoS* degradation and the *SLA* violations that any affected *VM* may experience. For example, we should limit the network resources allocated to a migration task that, due to its network bandwidth consumption, leads to an *SLA* violation for a *VM* hosted on the migration's target *PM*.
- limit the effects in the operation of a migrating *VM*. For example, a migration's duration can last a lot more than estimated, when a migrating *VM* constantly writes on blocks that need to be re-transferred to the target *PM*. An efficient real–time scheduler should be as non-interventional as possible with such a migrating *VM* and, still, should not let the migration extend beyond the time window.
- complete each task within the respective pre-defined time window.

Live migration requires that both the source and target *PM*s (used for hosting the *VM*) have access to the *VM*'s virtual disk(s). This requirement can be readily fulfilled for small clouds that use a shared common storage substrate. However, when it comes to large *IaaS* clouds built around the *shared-nothing* approach or clouds that have to interoperate across broadband networks, a common storage option is infeasible. In such settings, live migration can be attained through on-demand copy and synchronization of virtual disks across physical nodes.

The approach we propose in this paper, employs *on-demand virtual disk synchronization* to accommodate large numbers of simultaneous migration tasks. Our prime objective is to complete each migration within the designated time-constraint

This work has been partially supported by *i-Marine* and *Sucre* EU FP7 projects as well as ERC Starting Grant # 279237.

while not depleting disk and network resources. We introduce a novel policy that effectively appropriates resources to carry out the simultaneous migrations. To this end, we seek to accomplish near real-time load balancing while at the same time diminishing the performance penalties that migrations inevitably inflict. Prior work [4], [7]–[9] has shown the feasibility of synchronizing individual virtual disks and these results constitute the foundation for our proposal.

Our proposed resource management technique helps synchronize in a timely fashion virtual disk images across *PM*s and avoid counterproductive movements during periods of high-use of the underlying cloud resources. The low-level features used by our approach primarily deal with the consumption of *PM*-resources manifested as I/O and network throughput rates. These tunable features are captured in the context of our *MigrateFS* file system that runs on each *PM*. Instances of *MigrateFS* communicate over the network and jointly control the transfer of a virtual disk image between any two *PM*s. Based on input provided by performance monitoring tools [10], [11], we continuously adjust the following two rates during disk shipment: *a)* disk throughput available to the *VM*'s internal processes accessing the virtual disks under migration, and *b)* network throughput used for the purposes of migration. In this way, we are able to yield safe estimates on the exact time the migration will finish within the shared-nothing *IaaS* cloud. Estimates also help us delay migrations when the cloud experiences heavy workloads.

We develop our approach around a coordinating *Migrations Scheduler* and a distributed *network of Brokers* as we target not only intra- but also inter-cloud operations. The resource allocation policy followed by the *Brokers* allows for prioritization of migration tasks while taking into account the network status so that "hot" physical network links are not further stressed by virtual disk shipments. *Broker* policies drive the operation of *MigrateFS* in restricting the network as well as disk throughput. Our evaluation, based on both a *MigrateFS* prototype and simulation of large infrastructures, shows up to 24% less stress on saturated *PM*s during migration. It is also worth pointing out that we achieve these gains with minimal administrative effort from the cloud provider. The paper is organized as follows: Sections II and III present the salient features of *MigrateFS*. Section IV outlines our two disk–shipment management policies and Section V reports on our experimentation. Related work and concluding remarks are found in Sections VI and VII.

II. Overview of Our Approach

Live *VM* migrations consume considerable amounts of cloud resources and inflict heavy performance penalties on the migrating *VM*s. More importantly, SLAs offered by the cloud provider may fail as network and disk bandwidth get depleted. By placing time constraints on the migration tasks, the cloud administration can schedule the performance degradation during periods of low demand. For instance, a *VM* migration may commence during the night time and can be forced to complete before consumers start using their *VM*s again later in the day.

Policies [1], [2], [12] that help determine whether and when a *VM* should migrate, typically consider the average and current load in both *PM*s and *VM*s, projections on future *VM* resource consumption, and the SLAs to be satisfied. In our work, each migration task references *a)* the *VM* to be migrated, *b)* the source and target *PM*s involved, and *c)* the time period within which the migration has to complete. A load-balancer may need to predict when *VM*s are to be used by cloud consumers so that shipments of *VM*s across the network take place during low activity periods.

Our approach assumes the existence of a queue where all migration tasks arrive. We attempt to manage resources in a way that all migrations complete within their respective time-constraints while not failing the offered SLAs. Figure 1 shows the key components of an *IaaS*-cloud that our approach deploys. At the top of Figure 1 lays the *Cloud Middleware* such as OpenStack or OpenNebula [13], [14]. *Migration Tasks* produced by a *Load Balancing Policy* [1], [2], [12] in the context of the middleware are dispatched to the underlying physical infrastructure. We present three physical systems, each one featuring its own local physical disk and a *VM* hypervisor [15], [16]. The *VM*s hosted on each system place their data on virtual disks stored as files on physical disks. All physical systems communicate through a networking layer represented as a switch/router at the bottom of Figure 1. Our approach entails three components: the *Migrations Scheduler*, the *Brokers*, and a special-purpose file-system *MigrateFS*. *MigrateFS* offers resource management facilities -denoted as *Disk* and *Network Throughput Control Points* in Figure 1- exploited by the *Brokers* during *VM* migration.

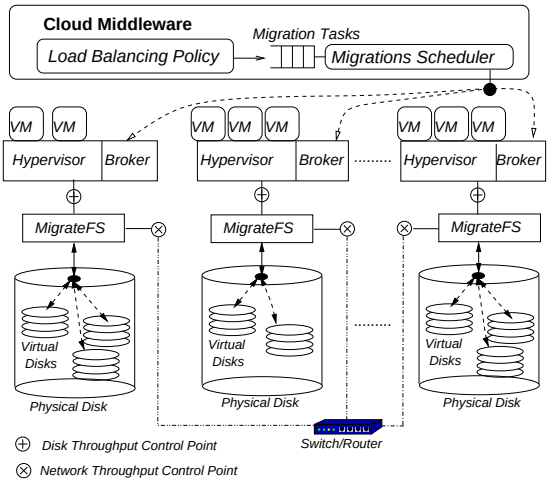

Fig. 1. High level view of our approach.

A. The Migrations Scheduler and its Brokers

The *Migrations Scheduler* takes as input a number of migration tasks along with their respective time-constraints. Tasks can be prioritized based on the cost of violating their time-constraints. A distributed network of *Brokers* oversees the resource consumption for the transfer of the *VM*s' disk

images. A low cost communication policy is used for the interaction between *Brokers* so as to ensure the scalability of our approach. As soon as the virtual disks are transferred across the *PM*s, the *Migrations Scheduler* contacts the hypervisor and initiates the live-migration of the *VM*. The synergy of the *Migrations Scheduler* and the *Brokers* ensures the timely movement of all *VM*s pending action.

In order to honor the migration time constraints, while respecting the SLAs offered, the *Brokers* have to manage two types of resources. First, the network bandwidth consumed for transferring virtual disk images must not hamper the performance of other, already on-line, *VM*s. Second, the virtual disk I/O bandwidth (available to the migrating *VM*) must be limited since this disk I/O translates to dirty disk pages that ultimately have to be transferred over the network. The usage of both resources (disk and network) can be constrained through the facilities offered by *MigrateFS*. The consumption of these two resources is continuously adjusted so that no SLAs fail in the dynamic cloud environment where the migrations take place.

B. MigrateFS and Resource Consumption Restrictions

Constraining resource consumption, so as to comply with time restrictions on migration tasks, has to be assisted by low-level cloud facilities. Such facilities must function outside the *VM*s as the abstractions enforced by the cloud hide the *VM*'s iternal operations from the cloud administration. In our approach, migration is assisted by a special purpose file system, *MigrateFS*, that *traps all I/O operations* targeting the virtual disk images of the migrating *VM*s. *MigrateFS* introduces a layer between the *VM* Monitor (*VMM*)/hypervisor and the physical device where the *VM*'s virtual disks are stored (Figure 1). Instances of *MigrateFS* collaborate in moving *VM* disk images without interrupting the operation of the *VM*s. During migration, blocks of the migrating *VM*'s disk images are copied to the target *PM*. When all blocks are transferred, the two copies of virtual disk images -in both the source and target *PM*s- are kept synchronized while the *VM* hypervisor completes the migration task by moving RAM contents and devices' states. It is in the context of *MigrateFS*, where the bandwidth of disk and network, are placed under restrictions.

- *Disk bandwidth:* As *MigrateFS* transfers blocks from the source to the target *PM*, the still on-line *VM* may write over already transferred blocks. These dirty blocks need to be transferred again. If the rate at which virtual disk blocks get dirty is higher than the rate blocks are transferred through the network, the migration task will not finish. Yet, the migration task must come to its completion under certain time constraints. To this end, an authorized *Broker* may contact *MigrateFS* and limit the rate at which the *VM* writes to its virtual disk. The decision on when and if such an I/O rate must be limited is based on the current network and disk I/O rates reported by *MigrateFS*.
- *Network bandwidth:* We need to ensure that no migration task will deplete the network resources of the *IaaS*-cloud. *MigrateFS* enables the *Brokers* to limit the network bandwidth consumed during migration. In this way, saturated network links, shared among migrating and non-migrating *VM*s, do not cause network SLA failures. With this approach, we are able to distribute resource consumption throughout the entire migration's time window.

III. Operational Aspects of *MigrateFS*

The I/O operations trapped by the *MigrateFS* layer can be replayed on remote *PM*s so as to create and maintain synchronized copies of virtual disk images. To this end, an instance of *MigrateFS* has to be installed on each *PM* that plays the role of either the source or target hosting node in a migration.

Each *MigrateFS* instance listens on a port for connections from either three sources: *a)* the *Migrations Scheduler* requesting the transfer of a *VM*, *b)* another *MigrateFS* instance sending data blocks and remotely replaying I/O operations, or *c)* a *Broker* overseeing the resource consumption of a migration task. *MigrateFS* presents an interface through which a *Broker* can query the progress of a migration task and set limits on the disk and network bandwidth. Table I summarizes the functionality *MigrateFS* offers for handling and monitoring a migration task.

TABLE I
MigrateFS NETWORK API FOR A *VM* MIGRATION TASK

Operation	Input/Output
Start a Migration	**In:** Target *PM*, disk image
Set Network Limit	**In:** Bandwidth in KB/sec
Set Disk Limit	**In:** Bandwidth in KB/sec
Query Progress	**In:** Selectively query the Network or Disk rate **Out:** The respective bandwidth in KB/sec
Query Completion	**Out:** True or False

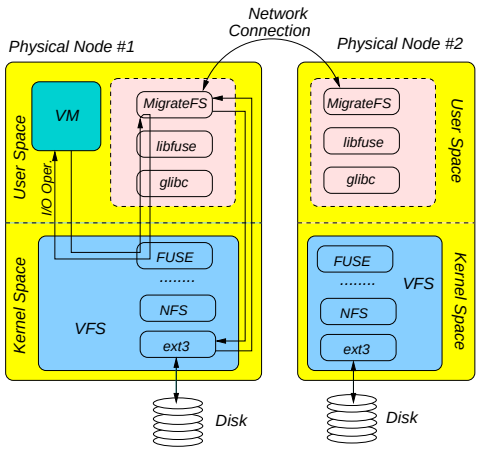

Fig. 2. Routing I/O calls through the layers of our approach.

As we show in Figure 2, *MigrateFS* is a user-space file system functioning as an intermediate between the *Linux* kernel and any underlying file system on the physical disk. *MigrateFS* is thus mounted over an already existing file system and mirrors its contents. The underlying file system is used to store virtual disk images. When an I/O request is issued by a process within a *VM* (I/O op. in Figure 2) it is forwarded through the virtual file system (VFS) API of the hypervisor to the file system mounted on the path where the virtual disks

are stored. The response to the I/O call is routed from the underlying file system through *MigrateFS* and the hypervisor to the *VM*. Our decision to mirror an already existing path in the physical disk storage greatly reduces the effort to set up *MigrateFS* on an already operational *IaaS*-cloud. An in-kernel file system implementation might yield higher performance but does so at the expense of a more intrusive and less deployable approach.

The main operation of *MigrateFS* is to synchronize a virtual disk image across any two *PM*s. To initiate a *VM* migration, the *Migrations Scheduler* has to contact the *MigrateFS* instance of the source *PM* where the *VM* is currently hosted and specify: a) the hostname of the target *PM*, b) the port on which the *MigrateFS* instance of the target *PM* listens, and c) the filenames of the disk images of the migrating *VM*. As soon as the request for a *VM* migration is received, the *MigrateFS* instance deployed on the source *PM* contacts the *MigrateFS* instance on the target and starts sending blocks of the virtual disk image over the network. Synchronizing a virtual disk across two *PM*s is a three phase process:

- *Phase 1:* Iterate once over all blocks of the virtual disk and send them to the target *PM*. As the *VM* remains online, in-*VM* processes may write over a portion of the blocks already transferred; these blocks are marked as dirty.
- *Phase 2:* Dirty blocks are transferred through the network. Here, the operating *VM* may continue to alter the content of blocks already transferred, thus rendering them dirty again. Sending blocks stops (phase 2 ends) as soon as all dirty blocks are transferred. For this to happen, the rate at which blocks get dirty must be less than the rate at which blocks are transferred through the network.
- *Phase 3:* During this phase, the virtual disks remain synchronized across the target and source *PM*s. Each write operation performed on the virtual disk residing in the source *PM* is replayed on the one residing on the target *PM*.

The purpose of this 3-phase copy of the virtual disks is to ameliorate the performance penalties of migration. The *VM* can operate with no restrictions during the first and second phases. In these two phases, the disk addresses of dirty blocks are kept in a thread-safe array protected through proper locking mechanisms. Dirty blocks are cleaned out during the second phase by sending them over the network. To reduce the high network latency penalty, the transfer block size (512 *KB*) is larger than the block size used by the local disk file systems. In phase 3, we have no dirty blocks. Each write operation is performed in both replicas of the virtual disk. During this phase, there is a significant impact on the I/O performance of the *VM* due to the network latency. Here, the block size is equal to the size of the I/O request. We expect the third phase to be short. During this phase, the hypervisor completes the live migration task by transferring the *VM*'s memory and devices' state. The details of moving the *VM*'s RAM and devices' state from the source to the target *PM* while minimizing the downtime are hypervisor-specific. With respect to the file system, the hypervisor sends a *sync* I/O system call as the last I/O operation right before the *VM* starts operating in the target *PM*. When the *sync* call is received, *MigrateFS* flushes all buffers (network and disk) so that the *VM* on the target *PM* will find the virtual disk in a consistent state.

IV. Resource Management for *VM* Migrations

Management of cloud resources expended during migrations ensures that *VM* movements are fulfilled within the specified time-constraints. Efficient resource handling should impose minimal overheads so that it can be applied to large, share-nothing clouds. In addition, resource handling should not impose special hardware requirements as the scalability of large infrastructures is based on the use of commodity hardware. We achieve the above through the distributed network of *Brokers* that interact using a low communication cost policy.

Pending *VM* migrations -produced by *VM* placement policies [1], [2], [12]- are delivered to the *Migrations Scheduler*. As soon as the latter decides that a migration task should start, it instantiates a *Broker* on the migration's source *PM*; the task of resource handling in shipping the *VM* is assigned to that *Broker*. The *Broker* acts so that Expression 1 remains true for the designated migration task.

$$\frac{Disk\_size}{(Net\_Rate - Dirty\_Rate)} + VMM\_time \leq finish\_time \quad (1)$$

The rate at which blocks get dirty (*Dirty_Rate*) as well as the rate at which blocks are moved (*Net_Rate*) from source to target *PM* are periodically queried from *MigrateFS*. The transfer rate must be, on average, greater than the dirty block rate so that the virtual disk network copy completes within the designated time frame (*finish_time*) and still leaves enough time (*VMM_time*) for the hypervisor to successfully complete the migration.

Priority-Based Resource Brokers

The network of *priority-based Brokers* requires two types of cloud operational information: *1)* Real-time notification of saturated network switches (hot-spots). Clouds facilitate monitoring tools [10] that detect stressed network links of the fixed (often tree-based) physical network topology [17]. *2)* The path of *VM* shipment. It is straightforward to compute such paths in a fixed network topology with static routing rules. The *Migrations Scheduler* computes migration paths and marks path sections shared among multiple migration tasks.

The *Migrations Scheduler* provides the *VM* shipment path upon the *Broker's* instantiation. The *Broker* registers for notifications on saturated switches to the corresponding cloud monitoring tools. As soon as the *Broker* projects that the designated migration time-constraint will be violated, it needs to request other *Brokers* to release (if possible) some of the network bandwidth they occupy. As the *Migrations Scheduler* is aware of all migrating *VM* disks sharing network paths any *Broker* can exploit this information to notify only those *Brokers* with which it shares saturated network links. Since *Brokers* exchange messages directly with each other in a peer-to-peer fashion there is no single message exchange hub. The distributed nature of *Broker* communication allows our approach to scale to the size of large cloud installations.

The cloud administration is allowed to specify which of the migrating tasks are important. *Brokers* responsible for such tasks should be the first to *a)* ignore network congestion and *b)* signal other *Brokers* to temporarily suspend their migration tasks whenever they face the danger of violating the migration's time frame. There are two thresholds, termed *danger* and *warning*, that are used in task prioritization. Both thresholds are percentages expressing the ratio between the time the migration has to be finished (*timetodeadline*) and the projected time the migration task will actually last (*timeleft*). Both threshold percentages are expected to be greater than 100% and the *danger* percentage to be less than the *warning* percentage as we first get a warning and then we face the danger of violating a constraint.

The operation of a *Broker* is described in Algs 1 and 2. Alg. 1 shows *when* the *Broker* chooses to limit the disk and network resources in the context of a single migration task, while Alg. 2 shows *how* this network limit is set. Network bandwidth consumption due to migration may cause other *VM*s to fail to uphold their SLAs. To address failing network SLAs, we should limit the network resources used by the migration process. When limiting the disk transfer rate, we ensure that SLAs will not fail due to the resources consumed for migration, but we do not commit to any migration time frame. Constraints on the migration time –given the available network bandwidth does not decrease– are honored when we limit the disk bandwidth available to the processes inside the *VM*. Limiting both transfer and disk rates enables us to offer *Deadline Scheduling* for the migration. *Deadline Scheduling* can be achieved even if we unbound the network transfer usage. In this case, we take the risk of failing network SLAs, yet, we potentially reduce the migration time.

Alg. 1 estimates the *timeleft* and the *timetodeadline* (lines 3 and 4) based on the disk blocks (*task.diskleft*) and two rates (*diskrate* and *netrate*) queried through the *Query Progress* call of *MigrateFS* (Table I). Should a warning of no compliance with a designated time-constraint be received (line 9), we limit the available disk bandwidth. In line 6 we assess the danger of failing to migrate the *VM* on time and if so we limit the used network rate (*setNetworkLimit* call in line 7). Similarly, in line 9, we get a warning of violating a time constraint if the ratio of *timeleft* to *timetodeadline* is greater than the *warning*. The higher the values of the *danger* and *warning* parameters, the sooner our algorithm will take action to secure the time constraints.

Limiting the network bandwidth is not based only on the *danger* threshold. We also limit the network usage rate if we detect a network contention, as indicated by the *backoff* flag. This flag is set to true under two conditions: *a)* the migration process causes a network SLA failure of a running *VM*, *b)* the network bandwidth consumed should be given to another migration task that is about to violate its time constraint. The *shouldBackOff* function detects saturated network links on the path between the source and target hosting *PM*s.

Algorithm 2 depicts how this network limit is set. The *Broker* takes into account the current network rate available through the input parameter *task*, the *backoff* flag and the

Algorithm 1 PriorityBasedMigrationsManagement

Input: *task*: The migration operation
period: Time between monitoring iterations
danger: Threshold indicating high chances of loosing the migration deadline
warning: Threshold indicating miss of the migration deadline

```
1: while (task.completed == false) do
2:    sleep(period);
3:    timeleft := computeTimeLeft(task.diskleft, task.netrate, task.diskrate);
4:    timetodeadline := task.deadline - now;
5:    backoff := shouldBackOff(task.source, task.target);
6:    if ((timetodeadline/timeleft ≥ danger) and (backoff == true)) then
7:       setNetworkLimit(backoff, task);
8:    end if
9:    if timetodeadline/timeleft ≤ warning then
10:      setDiskLimit(task);
11:   end if
12: end while
```

number of consecutive periods with no backoff request indicating no network congestion. Inspired by TCP, our approach reduces the limit of the network bandwidth usage by dividing the current network rate by two and increases the network limit linearly. Two factors influence our decision: the last decision to increase or decrease the network limit (outer **if-then-else** statement), and any request for releasing bandwidth made through the *backoff* flag. If we had limited the network usage and we still observe network congestion (**if** statement in line 2), we use the *task.netrate* divided by two as the new limit. If we had previously reduced the network rate and we now have no back off request, we mark the current network rate as one that causes no network congestion (line 5). This mark, stored in *task.lastOKnetlimit*, is used in line 10 where we have previously increased our network limit and we just got a back off request. In hope that our migration task caused the network congestion we quickly revert back to a network rate that did not cause any congestion in a previous period. Further *backoff* requests will cause lowering even more the network limit. The **else** clause of lines 11 to 18 handles the case where we continuously increase our network limit. We raise the network limit linearly, yet, there might be the case that a lot of bandwidth has become unexpectedly available due to an event that we are not informed of (e.g., another migration task has just finished). In this case, we need to probe the network availability and quickly take advantage of the extra bandwidth (lines 13 to 15).

Alg. 2 implements a policy that requires no communication with the other consumers of the network bandwidth. Priority is given to *VM*s failing their SLAs and to migration tasks in danger of violating their time constraints. In this context, we opt for a low-cost communication policy among *Brokers* as we target large cloud infrastructures. *Brokers* need only to announce the danger of violating the time constraint of a *VM* migration to a well specified subset of other *Brokers* so that the *shouldBackOff* call yields valid back-off requests.

V. Evaluation

We evaluate our approach in two ways. First, we use the *MigrateFS* prototype to quantify the overheads involved in hosting *VM*s in our file system. For this, we use a real

Algorithm 2 setNetworkLimit

Input: *task*: The migration task
backoff: True, if we are to reduce the network bandwidth, False otherwise
n: # of consecutive periods with no *backoff* request indicating no network contention
C: Step of network rate increase, in MB
Output: The network limit

```
1: if task.lastAction == "reduce bandwidth" then
2:    if backoff == true then
3:       return task.netrate / 2;
4:    else
5:       task.lastOKnetlimit := op.netrate
6:       return task.netrate;
7:    end if
8: else if task.lastAction == "increase bandwidth" then
9:    if backoff == true then
10:      return task.lastOKnetlimit;
11:   else
12:      task.lastOKnetlimit := task.netrate
13:      if consecutivePeriodsWithoutBackOff(task,backoff) > n  then
14:         return +∞ /*no network limit*/
15:      else
16:         return task.netrate + C
17:      end if
18:   end if
19: end if
```

cloud infrastructure setup in our lab using *Xen* 3.2-1 [16] and OpenNebula [14]. Second, we simulate large infrastructures and show the effectiveness of combining *MigrateFS* with our resource management polic. We implemented our own cloud simulator (in *Java*) that uses the proposed approach to appoint resources for *VM* migration.

A. MigrateFS Overheads

The *MigrateFS* prototype is implemented in *C* using *FUSE* [18] and *pthreads*. As source and target hosting nodes of migrating *VM*s we use two physical systems connected with a 1 *Gbps* Ethernet switch. Each physical node is equipped with 8 *GB* of RAM and an Intel(R) Xeon(R) CPU X3220 at 2.40GHz. The *VM* disk images are stored in an *ext3* file system.

File system benchmarking during normal operation – no migration: Using the *Bonnie++* [19] benchmark, we measure the performance overhead introduced by the additional layer of *MigrateFS*. The benchmark is executed within a paravirtualized *VM* featuring 2 *GB* of RAM and a single CPU core. We compare *MigrateFS* against three methods of accessing the virtual disk:

- *Local:* This method routes I/O system calls through the hypervisor's kernel directly to the file system (*ext3*) where the virtual disk image resides.
- *GlusterFS:* Virtual disks are stored in a distributed file system setup with *GlusterFS* [20]. *GlusterFS* is configured to use two nodes in RAID-1 configuration. With this setup, each write operation is performed on the local file system and on the remote node.
- *Mirror:* This method routes I/Os through the hypervisor to a FUSE file system. The FUSE file system traps and relays all I/O operations to the underlying file system holding the virtual disk images. In this manner, we quatify the FUSE-introduced overehad.

Figure 3 shows the read, write, and write-after-read, performance of *MigrateFS* as far as block I/O operations are concerned. When no migration operations are involved, the performance of *MigrateFS* is almost identical to the one we measure for the *Mirror* file system. This shows that the main overheads involved are introduced by the FUSE layer[1]. The write performance of *GlusterFS* is significantly hampered by the network latency as all files are replicated in a remote physical node (RAID-1 configuration). In this type of I/O call, *MigrateFS* proves superior. In the case of read operations, *GlusterFS* uses caching at the expense of RAM consumption and thus, it surpasses *MigrateFS*.

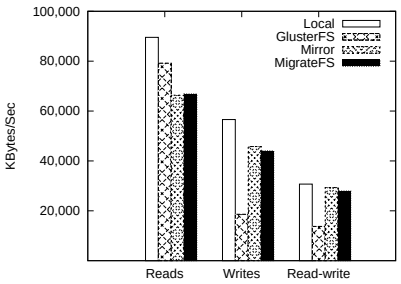

Fig. 3. Comparing block I/O performance of *MigrateFS* to other disk access methods.

B. Priority Based Management of Migration Tasks

Impact of scheduling migrations on SLA failures: The physical infrastructure we simulate in this experiment is made of 500 *PM*s. Each *PM* features a 10 *Gbps* network link to all other physical nodes. The *VM*s hosted by this physical infrastructure are evenly distributed among the *PM*s. Each *VM* has a 50 *GB* virtual disk and writes into it at a rate of 30 *MB/sec*. This 50 *GB* virtual disk has to be transferred during migration. Regarding network usage, there are no restrictions on the bandwidth consumed by each *VM*.

The evaluation scenario we present here consists of 1,000 migration tasks. We trigger one migration operation every 100 seconds. For the disk transfer operation we have the 10 *Gbps* of the *PM* network links at our disposal. However, the operating *VM*s need to use some of the available bandwidth for their normal operation. As we cannot account for the workload of all *VM*s collocated with the migrating ones, we set the available network bandwidth to follow a Gaussian distribution.

Consuming network resources for migrating *VM*s will stress the cloud resources. To quantify the effectiveness of our approach we use two metrics:

- *SLA violations* occur due to network bandwidth shortage. Bandwidth shortage may occur on any randomly selected *PM* as we cannot predict the behavior of each *VM* running. The duration of the resource shortage is set to 300 seconds (five 60 second periods). This shortage (marking an *SLA violation*) is extended for additional periods in case the migration operations consume the

[1]The performance of FUSE has improved in versions newer than the one used, yet, these improvements were not available in our testing environment.

entire 10 *Gbps* bandwidth available to the *PM* under stress.

- *The load of the network* is represented by the frequency of violations. That frequency is expressed as the percentage of *PM*s where a shortage occurs within a 60 second period.

Increasing the *load of the network* causes more *SLA violations*. Our goal is to limit the network utilization over stressed links used during migration.

We show how our resource management approach handles migrations with different time constraints. We compare our approach against the currently available cloud setup, denoted as "No Scheduling", where there are no constraints on the resources consumed. The *danger* and *warning* thresholds of Alg. 1 are set to 300% and 400% respectively. The network and disk limits are updated once every 60 seconds (the *period* in Alg. 1).

In Figure 4, we present our policy operating under three different migration completion time values, 200, 350 and 800 seconds, and the "No Scheduling" approach. We measure the periods (*y*-axis) we observe network SLA violations as we gradually increase the network load from 5% to 25%. High values indicate that we further stress the already limited network resource. The 25% load is an extreme case where in each minute (60 second period), 25% of the *PM*s links fail to satisfy the *VM*'s SLAs and they continue to display high load for the next five minutes.

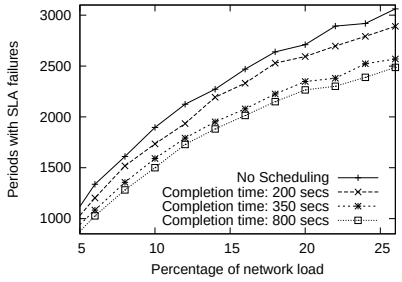

Fig. 4. Evaluating our approach under different time constraints set on migrations.

This experiment shows an important aspect of our work: by limiting the network and disk bandwidth, our policy waits for the "hot" spots (*PM*s) of the infrastructure to cool down before performing the migration. Reducing the acceptable migration time, forces the scheduler to use "hot" *PM*s instead of waiting.

Impact of prioritizing migration tasks on throughput: The *danger* and *warning* thresholds are used to prioritize migration tasks as they reflect when our resource sharing policy will take action to ensure a timely migration. In this experiment, we have three migration tasks, each one with a different set of *danger* and *warning* thresholds. The *danger* thresholds are 200%, 300%, 400% and the *warning* thresholds are 250%, 350%, 450% respectively for the *VM*s with IDs 0, 1, 2. All migration tasks have the same source and target *PM*s and thus use the same network links and share the same network bandwidth.

In Figure 5 we present the network throughput consumed by each migration task over time. All three tasks start in a danger state according to the throughput they have available and their *danger* threshold. In the period of 500 to 1,700 seconds first *VM*-0, then *VM*-1 and finally *VM*-2 exit the *danger* "zone" leaving more bandwidth available. A time period of equal bandwidth share follows (1,700 to 2,200). After that the migration tasks finish in the order their thresholds dictate. We do not see the remaining tasks take up the available bandwidth immediately after a migration finishes because as a migration nears its end, it often enters the danger state leading the rest of the migrations to release some of the network resources they occupy. This is because in the third phase of the migration, the network bandwidth consumed is affected by the short block size of the I/O requests.

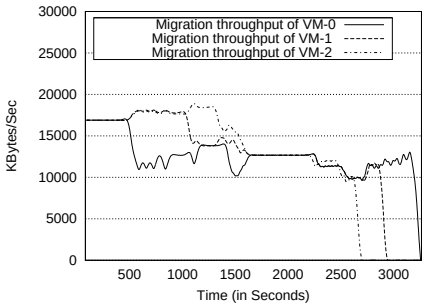

Fig. 5. Prioritize migration requests.

VI. Related Work

Live OS migration was shown [5] to have the potential to reduce downtime to tens of milliseconds by first copying RAM content and only after the *VM* state including virtual devices (e.g., CPU, network). By transferring the *VM*'s state in the early stages of the migration, *VM* post-copy [21] was shown to reduce the total number of RAM pages moved as no dirty pages are involved. The *Slowdown Scheduling Algorithm (SSA)* [6] seeks to reduce dirty page rate in accordance to the CPU activity of the migrating *VM*.

VM-Monitors (*VMMs*) [15], [16] are not concerned with migrating the *VM* persistence layer (i.e., the "virtual disk(s)"). In this direction, the cloud design may actually assist and its administration can explicitly replicate disk images at the block level [22], [23]. For small to medium-size clouds, designers and administrators can resort to SANs or deploy distributed file systems [20] (*DFS*s). The incremental transfer of virtual systems to different locations presents an alternative that avoids scalability issues often hindering *DFS*s. In [8], [9], live *VM* migration in WANs is advocated as a way to relieve overloaded *PM*s. A workload-driven migration of virtual storage is discussed in [24]. *VMware ESX 5.0* [4] is capable of live migrating *VM*s including the persistence layer using *IO Mirroring*; here, all write operations are performed concurrently in both source and target *PM*s during a *VM* migration.

Autonomic and load-balancing systems are often employed to schedule *VM* migrations. In such systems [1], [12], [25]–

[27] the performance of the cloud is continuously monitored and proper action is taken to handle bottlenecks and resource shortages. Our approach can work in tandem with such autonomic systems and can extend their effectiveness. Time-warranties are typically used to enhance the quality of service offered by real-time systems [28], [29]. I/O throttling during *VM* migration enables for time and performance warranties and ultimately for migration completion. *VMware ESX 5.0 IO Mirroring* [4] reduces the transfer rate to the slowest medium of the source or target *PM* and guarantees migration convergence. In [30], [31] live migration is applied to databases. In [32] data flow deadlines are accommodated through a network control protocol; this diversity of data flows calls for significant changes in the underlying infrastructure.

VII. Conclusions - Future Work

Live migrating *VM*s is of key importance to *IaaS*-clouds as it helps accomplish major operational and administrative objectives including effective load-sharing and improved utilization of physical machinery. However, the movement of *VM*s over the network inadvertently consumes significant cloud resources. In this work, we focus on emerging highly-scalable share-nothing cloud installations and employ on-demand virtual disk synchronization across *PM*s to attain live migration under explicit time-constraints. Based on network elements and computing systems' usage rates, our proposal accommodates all scheduled *VM* migrations within their specified time-frame by considering the dynamics of the underlying share-nothing infrastructure. Our approach is empowered by the combined use of a network of *Brokers* and the *MigrateFS* file system. *MigrateFS* effectively synchronizes disk images between physical computing systems, while the *Brokers* manage the resources of the share-nothing cloud elements. Our proposed resource management policy dynamically adjusts the bandwidth of both virtual disks and network consumed during the transfer of *VM* disk images. Our prototype experimentation demonstrates the minimal overheads involved in the operation of our approach and shows that we can honor the time-constraints set for migrations while significantly reducing SLA violations due to heavy network traffic. We plan to examine statistics-driven and cost aware *VM* migration scheduling algorithms in large infrastructures possibly consisting of interoperating *IaaS*-clouds and ivestigate the application of our approach to *PaaS*-clouds for attaining real-time migrations.

References

[1] C. Weng, M. Li, Z. Wang, and X. Lu, "Automatic Performance Tuning for the Virtualized Cluster System," in *Proc. of the 29th IEEE Int. Conf. on Distributed Computing Systems*, Montreal, Canada, June 2009.

[2] VMware, "VMware DRS - Dynamic Scheduling of System Resources," www.vmware.com/products/drs/overview.html, Oct. 2009.

[3] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation," in *Proc. of the 1st Int. Conf. on Cloud Computing (CloudCom'09)*, Beijing, China, Dec. 2009.

[4] A. Mashtizadeh, E. Celebi, T. Garfinkel, and M. Cai, "The Design and Evolution of Live Storage Migration in VMware ESX," in *Proc. of the 2011 USENIX Annual Technical Conference*, Portland, OR, 2011.

[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *Proc. of the 2nd Symposium on Networked Systems Design & Implementation*, Boston, MA, May 2005.

[6] Z. Liu, W. Qu, W. Liu, and K. Li, "Xen Live Migration with Slowdown Scheduling Algorithm," in *Proc. of the 2010 Int. Conf. on Parallel and Distributed Computing, Applications and Technologies*, ser. PDCAT '10, Wuhan, China, Dec. 2010, pp. 215–221.

[7] Y. Luo, B. Zhang, X. Wang, Z. Wang, Y. Sun, and H. Chen, "Live and Incremental Whole-System Migration of Virtual Machines Using Block-Bitmap," in *Proc. of IEEE Int. Conf. on Cluster Computing*, Tsukuba, Japan, September 2008.

[8] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live Wide-Area Migration of Virtual Machines Including Local Persistent State," in *In VEE '07: Proc. of the 3rd Int. Conf. on Virtual Execution Environments*, San Diego, CA, June 2007.

[9] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe, "CloudNet: Dynamic Pooling of Cloud Resources by Live WAN Migration of Virtual Machines," *SIGPLAN Not.*, pp. 121–132, March 2011.

[10] D. Josephsen, *Building a Monitoring Infrastructure with Nagios*. Upper Saddle River, NJ: Prentice Hall PTR, 2007.

[11] VMware, "vSphere," www.vmware.com/products, May 2012.

[12] P. Pradeep, H. Kai-Yuan, S. K. G., Z. Xiaoyun, U. Mustafa, W. Zhikui, S. Sharad, and M. Arif, "Automated Control of Multiple Virtualized Resources," in *Proc. of the 4th ACM European Conf. on Computer Systems*, Nuremberg, Germany, 2009.

[13] OpenStack, "www.openstack.org," May 2012.

[14] OpenNebula, "www.opennebula.org," May 2012.

[15] Kernel Based Virtual Machine, "www.linux-kvm.org," May 2012.

[16] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *SOSP*, New York, NY, 2003, pp. 164–177.

[17] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proc. of the ACM SIGCOMM Conf.*, Seattle, WA, August 2008.

[18] FUSE, "Filesystem in Userspace," fuse.sourceforge.net, May 2012.

[19] R. Coker, "Bonnie++ file system benchmark," www.coker.com.au/bonnie++, May 2012.

[20] Red Hat, "GlusterFS," www.gluster.org, May 2012.

[21] M. Hines and K. Gopalan, "Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning," in *Proc. of ACM SIGPLAN/SIGOPS Int. Conf. on Virtual Execution Environments (VEE'09)*, Washington, DC, March 2009, pp. 51–60.

[22] "Disrtibuted Replicated Block Device," www.drbd.org, Sept. 2012.

[23] D. T. Meyer and B. Cully, "Block Mason," in *Proc. of the First Workshop on I/O Virtualization*, ser. WIOV '08, San Diego, CA, Dec. 2008.

[24] J. Zheng, T. Ng, and K. Sripanidkulchai, "Workload-Aware Live Storage Migration for Clouds," in *Proc. of the 7th ACM Int. Conf. on Virtual Execution Environments*, Newport Beach, CA, 2011.

[25] A. Danak and S. Mannor, "Resource Allocation with Supply Adjustment in Distributed Computing Systems," in *Proc. of the 30th IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, Genoa, Italy, June 2010.

[26] F. Hermenier, X. Lorca, J. Menaud, G. Muller, and J. Lawall, "Entropy: a Consolidation Manager for Clusters," in *Proc. of ACM SIGPLAN/SIGOPS Int. Conf. on VEEs*, Washington, DC, March 2009.

[27] K. Tsakalozos, M. Roussopoulos, V. Floros, and A. Delis, "Nefeli: Hint-based Execution of Workloads in Clouds ," in *Proc. of 30th IEEE Int. Conf. Distributed Computing Sytems (ICDCS)*, Genoa, Italy, June 2010.

[28] S. Biyabani, J. Stankovic, and K. Ramamritham, "The Integration of Deadline and Criticalness in Hard Real–Time Scheduling," in *Proc. of the Real–Time Systems Symposium*, Huntsville, AL, Dec. 1988.

[29] J. Bruno, J. Brustoloni, E. Gabber, B. Ozden, and A. Silberschatz, "Disk Scheduling with Quality of Service Guarantees," in *Proc. of the IEEE ICMCS*, Florence, Italy, June 1999, pp. 400–405.

[30] S. Barker, Y. Chi, H. J. Moon, H. Hacigümüş, and P. Shenoy, ""Cut Me Some Slack": Latency-Aware Live Migration for Databases," in *Proc. of the 15th EDBT*, Berlin, Germany, 2012.

[31] A. J. Elmore, S. Das, D. Agrawal, and A. El Abbadi, "Zephyr: Live Migration in Shared Nothing Databases for Elastic Cloud Platforms," in *Proc. of the 2011 ACM SIGMOD Int. Conf. on Management of Data*, ser. SIGMOD '11, Athens, Greece, 2011, pp. 301–312.

[32] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better Never than Late: Meeting Deadlines in Datacenter Networks," in *Proc. of the ACM SIGCOMM Conf.*, Toronto, Canada, Aug. 2011, pp. 50–61.