ISSN:0975-9646

Civilizing Exploitation of Communication Clouds

Janga Santhosh, Puram Pradeep kumar, Majoju Sridhar kumar

Computer science and Eng. Department,
JNT University,Karimnagar,
Andhra Pradesh, India

***Abstract*— the advantage of infrastructure -as-a-Service (IaaS) clouds is providing users on-demand access to resources. to provide on-demand access, cloud providers must either significantly overprovision their communication or reject a large proportion of user requests (in which case the access is no longer on-demand). At the same time, not all users require truly on-demand access to resources. Many applications and workflows are designed for recoverable systems where interruptions in service are expected. For instance, many scientists utilize High Throughput Computing (HTC)-enabled resources, such as Condor, where jobs are dispatched to available resources and terminated when the resource is no longer available. We propose a cloud communication that combines on-demand allocation of resources with opportunistic provisioning of cycles from idle cloud nodes to other processes by deploying backfill Virtual Machines (VMs). We demonstrate that a shared communication between IaaS cloud providers and an HTC job management system can be highly beneficial to both the IaaS cloud provider and HTC users by increasing the exploitation of the cloud communication and contributing cycles that would otherwise be idle to processing HTC jobs.**

***Key words*: Cloud computing, Communication-as-a-Service, High Throughput Computing**

I. INTRODUCTION

In the current years, Infrastructure-as-a-Service (IaaS) cloud computing has emerged as an attractive alternative to the acquisition and management of physical resources. The on demand provisioning it supports allows users to elastically expand and contract the resource base available to them based on an immediate need – a pattern that enables a quick turnaround time when dealing with emergencies, working towards deadlines, or growing an institutional resource base. This pattern makes it convenient for institutions to configure private clouds that allow their users a seamless or near seamless transition to community or commercial clouds supporting compatible VM images and cloud interfaces. Such private clouds are typically configured using open source IaaS implementations such as Nimbus or Eucalyptus. However, such private cloud installations also face a exploitation problem. In order to ensure on-demand availability a provider needs to overprovision: keep a large proportion of nodes idle so that they can be used to satisfy an on-demand request, which could come at any time. The need to keep all these nodes idle leads to low exploitation. The only way to improve it is to keep fewer nodes idle. But this means potentially rejecting a higher proportion of requests – to a point at which a provider no longer provides on-demand computing. This situation is particularly hard to accept in the world of scientific computing where the use of batch schedulers typically ensures high exploitation and thus much better resource amortization. Thus, potential low exploitation constitutes a significant potential obstacle to the adoption of cloud computing in the scientific world. At the same time, while the on-demand capabilities of IaaS clouds are ideal for many scientific use cases, there are others that do not necessarily require on-demand access to resources. Many systems, specifically volunteer computing systems such as SETI@Home and Folding@Home , are capable of taking advantage of resources available opportunistically and are also preemptible, i.e., designed as failure resilient systems where interruptions in service can be handled without compromising the integrity of computation. One example in the scientific community is the use of high throughput computing (HTC), as implemented by e.g., the Condor system where users employ HTC-enabled resources to process their workloads. These applications are designed to "scavenge"unused resource cycles: for example, when a user stops using their desktop, the screensaver might use the resource to run a volunteer computing program. The job may then be pre-empted when the resource becomes unavailable (i.e., the user is using it again), in which case the job is typically requeued and rescheduled on another available resource by the HTC system that manages it. We propose a cloud communication that combines ondemand allocation of resources with opportunistic provisioning of cycles from idle cloud nodes to other processes, such as HTC, by deploying backfill VMs. Backfill VMs are deployed on idle cloud nodes and can be configured to perform any desired function. A backfill VM is terminated when the resource is needed to satisfy an on-demand request. If we can ensure that the computation occurring in backfill VMs is resilient to such sudden termination, the time that would otherwise be idle can be profitably spent. Furthermore, cycles via backfill VMs can be provided to users at a lower cost than on-demand VMs because of the cloud providers ability to terminate the instances when needed, thus for users that work with HTC resources and possibly expect such behavior already, backfill VMs would provide a less expensive option when moving their workloads to the cloud. Overall, this design achieves two goals: for cloud providers, it offers a path to higher utilized clouds; for cloud users, it offers another type of resource lease, potentially cheaper than on-demand, non preemptible resource. In our work we extend the Nimbus toolkit to deploy backfill VMs on idle Virtual Machine Monitor (VMM) nodes .

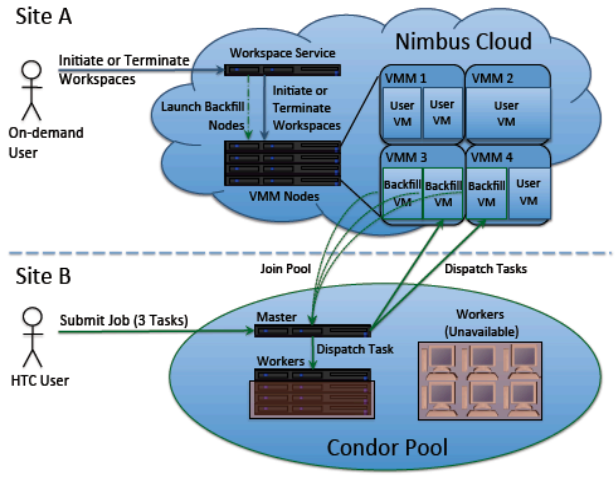

Figure 1. Example Backfill Deployment

Nimbus is an open source toolkit for deploying IaaS clouds, designed with extensibility in mind, which makes it particularly suitable for projects such as the one described here. To illustrate how the system works, we configure the backfill VMs as Condor workers that integrate with a Condor pool to process HTC jobs. We evaluate the ability of the system to increase exploitation of the IaaS cloud communication without sacrificing the ability of the IaaS cloud to provision resources on-demand. We also evaluate the ability of the system to contribute cycles that would otherwise be idle to processing HTC jobs. We find that during certain portions of our experimental evaluation backfill VMs contribute to an increase in the exploitation of the IaaS cloud communication from 37.5% to 100% with only 6.39% overhead cost for processing the HTC workload. Additionally, backfill VMs process the entire Condor workload using what would have otherwise been idle cycles. The remainder of the paper is organized as follows.

II. MOVE TOWARDS

A compute communication cloud operates by allowing a user to make *leases* against its pool of resources; an ommunication lease makes a resource available to the user based on set of lease terms defining the availability, capacity and general conditions of a lease. In our system we focus on investigating two types of leases: *On-demand, non-preemptible and flexible leases* give a user access to a resource within interactive time of making the request and make the resource available for an agreed-upon period of time. The user can deploy any VM compatible with the system. *Opportunistic, preemptible and pre-set leases* give a user access to a resource at an indeterminate time and make the resource available to the user for an indeterminate amount of time. Further, this resource is pre-defined for the user by the cloud administrator, i.e.the user cannot provide his or her own VM. In the rest of the paper we will be referring to them as *ondemand leases* and *preemptible leases* respectively. We define the following roles in our system. An *on-demand user* is a user that requests on-demand VMs/leases from an IaaS cloud. An on-demand *VM* is an IaaS VM that has been provisioned via on on-demand lease for a specific user. A *backfill VM* is a VM that has been deployed automatically by the IaaS cloud manager on an idle IaaS node using a preemptible lease. An *IaaS cloud dministrator* is the person or persons responsible for configuring and managing the IaaS cloud resource. An *HTC user* is a user that submits jobs to an HTC queue and an *HTC worker* is a system that process jobs from the HTC queue. In the context of IaaS clouds, backfill VMs are generic VMs deployed on IaaS resources using a preemptible lease that may be configured to perform any function. Backfill VMs have two major constraints. First, backfill VMs may be terminated suddenly in order to free up space for the IaaS cloud manager to service an on-demand lease. Second, because of the unpredictable timing of on-demand leases (from any number of unique users), a variable number of backfill VMs may be available at any given time. Thus, we assume that applications executing inside backfill VMs are designed to handle environments that contain a variable number of workers that may join or leave the system at any time. Certain applications are not well suited for these volatile environments, for example, parallel applications that require all processes to be present for the duration of the application's execution and lack checkpoint/restart capabilities. An example of a system that is well-suited for backfill VMs are High Throughput Computing (HTC) workloads . HTC workloads typically consist of a large number of jobs that eventually need to be processed. An assumption of HTC workloads is that they do not have an immediate deadline and therefore do not need resources to be available at a particular time. In addition, terminating individual HTC jobs in the middle of execution and requeuing them for later execution is an acceptable action as long as the system is eventually able to process the workload. This work further assumes that backfill VM images and deployments are managed by the IaaS cloud administrator; backfill VMs are not directly managed by remote cloud users. This work also assumes that the HTC scheduler is not aware of the IaaS cloud scheduler details and vice versa. Thus, the scheduling of HTC jobs on HTC workers (in backfill VMs) is separate from the launching and terminating of user VMs on the IaaS cloud.

A. Architecture

Figure 1 is a simple example deployment consisting of an IaaS Nimbus cloud using backfill VMs to run Condor HTC jobs in order to increase cloud communication exploitation. In this example the HTC user submits 3 individual tasks to the Condor master, which is able to schedule 1 task immediately on a local worker. However, because the remaining resources in site B's Condor pool are busy, Condor leverages the cycles provided by site A's backfill VMs to launch the other 2 tasks immediately. Condor is an ideal candidate for such a deployment because of its original design as a cycle-scavenger where idle desktop machines execute jobs until the system's user returns, after which the job is either migrated to another system or prematurely terminated and re-launched on another system. A backfill deployment could be much more complex then the example depicted in Figure 1. For instance, backfill VMs could host a number of different backfill processes in addition to Condor workers. The Condor workers could

also be configured to execute jobs from multiple Condor pools or sites instead of a single pool, as shown in the figure. IaaS cloud administrators must consider a variety of different backfill configurations when deploying a backfill solution on an IaaS cloud. The configuration is influenced by the characteristics of the underlying physical IaaS resources, the users of on-demand leases, and the users of preemptible leases. First, appropriate backfill applications and workflows should be identified. These applications should accommodate the constraints discussed previously, namely, they should be able to utilize a variable number of nodes that may join or leave the system at any time. Second, if the IaaS Virtual Machine Monitor (VMM) nodes have multiple cores, the IaaS cloud administrator must determine the granularity with which to deploy backfill VMs. One possible solution is to deploy a single backfill VM for each core on a VMM node. This approach allows the IaaS cloud manager to have fine-grained control over VM deployment. For example, a node with 8 cores may be comprised of 3 user VMs and 5 backfill VMs. However, the disadvantage of this approach is the increased overhead introduced by running additional VMs. An alternative solution is to deploy a single backfill VM per VMM node, utilizing all of the available cores in the backfill VM. This approach reduces the virtualization overhead on the VMM node since only a single backfill VM would be running, however, if the cloud receives an on-demand user request for a single core it is possible that the entire backfill VM will need to be terminated to satisfy the on-demand request. This would leave the additional VMM cores idle. Alternatively, the administrator may wish to configure VM deployments based on allocation of other resources, such as RAM, instead of (or in addition to) CPU cores. Third, the IaaS cloud administrator must determine the approximate size of the backfill deployment, relative to the size of the IaaS cloud. The administrator may allow the backfill deployment to utilize all available nodes, however, the administrator should consider the additional overhead required to terminate backfill nodes in order to fulfill on-demand user requests. Depending on the method used to terminate backfill VMs (e.g. immediately trash the VM or cleanly shut it down) and the number of backfill nodes being terminated, this overhead may be significant. Consequently, IaaS cloud administrators may configure backfill deployments to only utilize a percentage of idle nodes or never allow backfill deployments exceed a preset and static number of idle nodes. Finally, the IaaS cloud administrator must determine the backfill VM image deployment method. A fresh backfill VM image could potentially be transferred from the VM image repository for every backfill VM deployment (this process is referred to as VM image propagation and is a typical pattern that remote users expect for their deployments); however, this introduces network contention and may slow the deployment of on-demand user VMs. Another solution is to propagate the backfill image to the node and cache it on the node, only propagating it again if the image is updated or is removed from the cache on the node. A third and simple solution is to manually place the backfill image on each VMM node. Thus, when a backfill VM boots, the image is already on the VMM node and doesn't require an image to be copied. This approach reduces network contention (since it is only performed at select times) and reduces launch time for the VM (since the backfill image doesn't need to be copied across the network). However, any changes to the backfill image require that the IaaS cloud administrator push it out to all VMM nodes.

B. Backfill Termination Policies Backfill VMs are deployed on idle VMM nodes and terminated whenever space is needed to service an on-demand lease. However, the specific backfill VMs that are selected for termination may impact the services, applications, or workflows executing inside those VMs. For this reason, ideally the backfill VM termination policies should consider the applications and services running inside those VMs as well as generic factors such as the need for clean shutdown, the ability of an application to checkpoint/restart, etc. We discuss below two simple policies that do not integrate such hints and highlight their shortcomings. One simple policy is to select the backfill VMs to terminate at random. This approach ignores a number of factors that could impact the services and applications running inside backfill VMs. In particular, if the random backfill VM selected for termination is the only backfill VM performing a useful task then its work may be lost while idle backfill VMs continue running. Another policy is to select the backfill VM that has been running for the least amount of time. This policy makes the assumption that the backfill VMs running the longest also run long-running jobs that have performed the most work and therefore will lose the most work when terminated. The intuition behind this policy is that a workload consisting entirely of short running jobs (e.g. less then a few minutes) will only be slightly impacted by the termination of any backfill VM however, workloads that consist of long running jobs (or a combination of long and short jobs) will be impacted more by the termination of a VM that has been deployed for an extended period of time. In this case all of the VM's work will be lost unless the application supports checkpoint/restart or migration. It is, however, possible that the backfill VM running the longest will not always be the backfill VM with the most work to lose; without tighter integration between the HTC job manager and backfill termination policy this information is not readily accessible. More advanced backfill termination policies will require "hints" from the IaaS scheduler in the form of a more complete picture of cloud VM placement or integration with the applications and services running inside of the backfill VMs. For instance, if the backfill termination policy is aware of VM placement on individual VMM nodes, it may be able to select the fewest number of backfill VMs for termination in order to satisfy an on-demand lease instead of blindly terminating backfill VM's until the request can be fulfilled. Alternatively, if the backfill termination policy integrates with the applications and services running inside of backfill VMs then it may be able to identify which backfill VMs are performing useful tasks and avoid terminating those VMs.

III. IMPLEMENTATION

We extend the open source Nimbus cloud computing toolkit, which provides on-demand access to resources (in the form of VMs), to support the deployment of preemptible leases

on idle cloud nodes, also referred to as backfill. We make a number of simplifying assumptions in our current implementation. First, the Nimbus administrator must configure backfill. On-demand cloud users cannot elect to deploy backfill VMs. Second, the current implementation is capable of using only a single backfill VM image per VMM node. Different backfill VM images could be deployed on different VMM nodes, allowing multiple backfill VM images to operate within the same IaaS cloud, each performing different functions. Unless the user specifies the max number of backfill instances, backfill automatically attempts to deploy as many backfill VMs as possible when it is enabled. Initially, we support two termination policies for selecting backfill VMs for termination in order to fulfill an on-demand lease. The first policy simply selects a random backfill VM. The second policy, and the default, terminates the most recently deployed backfill VM in an attempt to minimize the amount of work "lost" by the premature termination of backfill VMs. In future work we hope to add additional backfill termination policies. Finally, our backfill implementation cleanly shuts down the backfill VM. Clean shutdown requires additional time over trashing the VM, however, performing a clean shutdown notifies services and applications running inside the backfill VM that the VM will be terminated, allowing them to respond appropriately (e.g. notify a central manager to reschedule currently running jobs).

A. *Backfill Configuration Options* Only the Nimbus cloud administrator can configure Backfill. The main configuration options are specified in a backfill.conf file on the Nimbus service node, allowing administrators to easily configure and deploy backfill VMs on Nimbus clouds. The backfill.conf options include:

- Backfill.disabled: This option specifies whether backfill is enabled or disabled for the specific cloud. The default is disabled.
- Max.instances: This option specifies the maximum number of backfill VMs to launch (assuming there are enough idle nodes). The default, 0, launches as many as possible.
- Disk.image: This option specifies the full path to the backfill VM image on the VMM nodes. This option assumes that the backfill VM image has already been pushed out to the VMM node, the Nimbus service does not automatically transfer it. The image must be in the same location on every VMM node.
- Memory.MB: This option specifies the amount of memory (RAM) to use for backfill VMs. The default is 64 MB.
- VCPUs: This option specifies the number of VCPUs to use for backfill VMs. The default is 1.
- Duration.seconds: This option specifies the amount of time (in seconds) backfill VMs should run before being terminated (currently Nimbus doesn't support "infinite" length VM deployments). The default is one week.
- Termination.policy: This option specifies the termination policy to use when terminating backfill VMs. The policies currently supported include a "most recent" policy that first terminates backfill VMs running for the least amount of time and an "any" policy that simply terminates a random backfill VM. The default is the most recent policy.
- Retry.period: This option specifies the duration (in seconds) that the backfill timer waits in between attempts to deploy backfill VMs on idle VMM nodes. The default is 300 seconds.
- Network: This option allows the IaaS cloud administrator to specify whether the backfill VMs should use the public network or private.

B. Extensions to the Nimbus Workspace Service We modified the Nimbus workspace service to support backfill VM deployments. The workspace service is responsible for managing the VMM nodes and servicing ondemand user requests for VMs. In particular, we added a backfill Java class that contains the majority of the backfill implementation code. The backfill configuration file is read when the Nimbus workspace service is started; if backfill is enabled then the service attempts to launch backfill VMs untilthe request for resources is denied or the maximum number of backfill instances is reached (as specified in backfill.conf). The service also starts a backfill timer that continually loops, sleeping for the duration specified by duration.seconds in backfill.conf, and attempts to launch backfill VMs when it wakes (until the request for resources is denied or the maximum number of backfill instances have been launched).

As part of our modifications to the Nimbus workspace service, we also modified its scheduler to detect any rejected requests for on-demand user VMs. If we detect a rejected ondemand user request and backfill is enabled, we attempt to terminate the appropriate number of backfill VMs so that the user request can be fulfilled. After the backfill VMs are terminated we attempt to service the user request again. If we are not able to service the request, we continue terminating backfill VMs until we are able to service the request or all backfill VMs are terminated. If all backfill VMs are terminated and we are still unable to service the request, then the request is rejected. We also modified the scheduler to detect when ondemand users are terminating VMs. In this case backfill attempts to re-launch backfill VMs (if backfill is enabled) without waiting for the backfill timer to expire. In general this is an acceptable approach, however, there is a design flaw in this initial implementation. It is possible that all backfill VMs could be terminated and yet the on-demand request could still be rejected. In this case the ideal solution would be to recognize, upfront, that the IaaS cloud is unable to fulfill the on-demand request and, therefore, the on-demand request should be rejected immediately before terminating any backfill VMs. However, recognizing this upfront requires a complete picture of the VM placement and the location of individual VM deployments on VMM nodes.

IV. EVALUATION

Our evaluation examines an IaaS backfill deployment from two perspectives. First, we consider the ability of the system to increase exploitation of the IaaS cloud communication without sacrificing the ability of the cloud to provision resources ondemand. Second, we consider the ability of the system to contribute otherwise idle cycles to

process HTC jobs using backfill VMs. We use Condor as the HTC job manager, leveraging its ability to requeue jobs that are interrupted during execution. For the evaluation we deploy a backfill-enabled version of Nimbus 2.6 on FutureGrid . Nimbus runs on a cluster of 16 VMM nodes with 2.4 GHz 8-core Intel Xeon processors and 24 GB of RAM with 20 GB allocated for user VMs, allowing for a total of 128 single-core VMs. The Nimbus workspace service node runs on an additional node. The Nimbus workspace service listens for incoming on-demand user requests for VMs and launches or terminates the VMs on the VMM nodes. This node also hosts the user VM image repository. In our experiments, we assume that a single backfill VM utilizes the entire VMM node (all 8 cores). We choose this level of granularity in order to reduce virtualization overhead for backfill VMs and avoid additional network contention caused by transferring a backfill VM image over the network each time it was deployed on an idle cloud node; instead the backfill VM images were manually copied to the VMM nodes before the evaluation began. Backfill VMs are configured as Condor worker nodes, preconfigured (at boot) to join our Condor master running on our evaluation node. The Condor pool does not contain any additional worker nodes. We use an additional two nodes (identical to the VMM nodes described above) to generate the workload. One node is used to host the Condor master and queues the Condor jobs. The second node executes the workspace service workload, requesting on-demand user VMs. On-demand user requests only request a single core. For all of the evaluations involving backfill we use the most recent backfill termination policy. The most recent backfill termination policy first terminates the backfill VMs that have been running for the least amount of time. The backfill VMs are terminated using clean shutdown. Cleanly shutting down backfill VMs enables the Condor workers running inside of the backfill VMs to notify the Condor master to reschedule its jobs. If clean shutdown is not used with Condor and the backfill VM is simply trashed, then Condor relies on timeouts before rescheduling jobs, which can take up to two hours. (As of the time of this writing Condor has an experimental feature to reverse the direction of its pings that determine the status of worker nodes, this would eliminate the long timeout period and the need to cleanly shutdown the backfill VMs. We enabled the feature, however, we did not observe the system behaving as expected. Interrupted jobs were still experiencing prohibitively long delays before being resubmitted to the Condor queue. Therefore, we did not use this feature for the evaluation, instead we terminate the backfill VMs using clean shutdown.)

For the evaluation we define the following metrics:

 Exploitation is the percentage of user cycles consumed by CPU cores on the VMM nodes in the IaaS cloud that are either running an HTC job or running an on-demand user VM. Because backfill launches VMs on any idle VMM node, regardless of the presence of HTC jobs, it is possible for the entire IaaS communication to be running backfill VMs on all VMM nodes but still have 0% exploitation. For our evaluation backfill VMs must be running Condor jobs for them to contribute to the overall exploitation of the communication.

 First queued time is the amount of time that elapses between the time when a Condor job is submitted and when it first begins executing.

 Last queued time is the amount of time that elapses between the time the Condor job is first submitted and the time the Condor job finally begins executing for the last time before completing successfully. We note that it is possible for backfill VMs to be terminated by the deployment of ondemand user VMs, preempting Condor jobs executing in backfill VMs, and thus requiring their resubmission. While this may happen to a Condor job any number of times, it is presumed that the job *User VM service response time* is the amount of time it takes the Nimbus service to respond to an on-demand user request, i.e., the time between when the service first receives the request and the time it determines whether a VM will be launched or that the request will be rejected. This time does not include the amount of time that it takes to actually boot the on-demand user VM or propagate the VM image, only the amount of time it takes the service to determine whether or not the request will be handled. If backfill is enabled and backfill VMs need to be terminated to deploy an on-demand user VM, the user VM service response time will include the necessary time to terminate backfill VMs.

A. Workload Traces The workloads we selected are based on real workload traces, modified to fit the size of our environment. The Condor workload used for the evaluation consists of a Condor trace from the Condor Log Analyzer at the University of Notre Dame . The workload contains 748 serial jobs that each sleep for differing amounts of time, with a minimum of 1 second, a maximum of 2089 seconds, and a standard deviation of 533.2. The Condor trace submits 400 jobs to the Condor queue immediately, followed by an additional 348 jobs 2573 seconds later. Along with the Condor workload we consider an on-demand IaaS cloud workload that we selected from the University of Chicago (UC) Nimbus science cloud [16]. We chose this particular workload trace because, despite its lack of dynamism, it is generally characteristic of the traces we observed on the UC Nimbus cloud. We did not observe the UC Nimbus cloud to be highly dynamic over relatively short time periods (e.g., a few hours). User requests were typically for a static set of instances over a long period of time (e.g. 6 VMs for 24 hours). In cases where user requests overlapped, the requests often overlapped for extended periods of time (e.g. 6 hours). Additionally, we selected this trace because it demonstrates the expected behavior of an overprovisioned cloud communication that is the focus of this work, i.e., it contains many idle VMM nodes available to service ondemand requests. Although there are an infinite number of possible on-demand and HTC workload scenarios that we could have generated for our evaluation, many which may have artificially highlighted the usefulness of backfill to either the on-demand user community or the HTC user community, we instead chose to base our evaluation off of two realistic workload traces. By choosing two realistic workload traces we are able to demonstrate and evaluate the usefulness of backfill to both

communities given at least one possible scenario. (Furthermore, we selected an on-demand trace from the considerably smaller UC Nimbus science cloud then a larger and possibly more dynamic cloud provider, such as Amazon or the Magellan cloud at Argonne National Laboratory , because of the lack of availability of such traces at the time of this work.) Because the University of Chicago Nimbus cloud only contains a total of 16 cores and our evaluation environment contains 128 cores we multiplied the workloads by 8 so that 16 individual requests for the University of Chicago cloud (16 cores) would be 128 individual requests for the entire 128 cores in the evaluation environment. Thus, an individual request for a single core on the University of Chicago cloud is 8 individual requests, each for a single core, in our evaluation environment. The on-demand user workload requests a total of 56 individual VMs over the course of the evaluation. Finally, we terminate the evaluation shortly after the overlapping Condor trace completes. Both workloads submit individual and independent requests; each request is for a single core. In the Condor workload the jobs simply consist of a program that sleeps for the desired amount of time. In the on-demand workload VMs are started and run for the appropriate duration. Backfill VMs are capable of executing 8 jobs concurrently across the 8 cores in a backfill VM, while individual on-demand user requests are single-core VMs. RAM is divided evenly among the VMs.

B. Understanding System Behavior

To understand the system behavior we compare three different scenarios. The first scenario only considers the ondemand user workload; the number of cores used in this workload is shown in Figure 2. In this case the IaaS cloud achieves an average exploitation of 36.36%, shown in Figure 5, with a minimum exploitation of 0% and a maximum exploitation of 43.75%. The second scenario simply involves running the Condor workload on all 16 VMMs (128 cores) without the on-demand user workload. In this case the entire Condor workload completes in approximately 84 minutes (5042 seconds), as shown in Figure 3. In the third scenario the Condor workload is overlaid with the on-demand user workload. The Condor workload takes an additional 11 minutes and 45 seconds over the case where Condor has exclusive access to the resources, completing in approximately 96 minutes (5747 seconds), as shown in Figure 4. However, the exploitation of the cloud communication, shown in Figure 6, increases to an average exploitation of 83.82% with a minimum exploitation of 0% and a maximum of 100%. As the Condor jobs complete (just before 6000 seconds in the evaluation) exploitation again drops because the IaaS cloud is no longer running Condor jobs in addition to on-demand user VMs. The large increase in exploitation is due to the fact that the cloud communication is no longer solely dedicated to servicing on-demand user VM requests, instead the cloud communication is also able to process jobs from a Condor workload without compromising its ability to service on-demand VM requests. The increase in exploitation is dependent upon the amount of work in the HTC workload. Naturally, longer and more HTC jobs will translate into higher exploitation. While increased exploitation certainly benefits the cloud provider, Figure 4 also demonstrates that it is advantageous to HTC workloads. The workload, which originally takes approximately 85 minutes on the same dedicated hardware (Figure 3), is only delayed by 11 minutes and 45 seconds (completing in just under 96 minutes) when on-demand user VMs are introduced into the system as shown in Figure 4. However, presumably the cost of utilizing backfill nodes would be lower than utilizing dedicated on-demand user VMs since backfill VMs may be reclaimed by the cloud provider without warning.

C. Understanding System Performance To understand how the IaaS cloud environment and backfill solution impacts on-demand users and HTC users we again consider the three different scenarios. The first scenario involves the on-demand user workload. The second scenario involves Condor jobs running on the 16 VMM nodes without interruption from on-demand user VMs and the third scenario overlays the first two. In Figure 7 we can see that the Condor first queued time is smallest when no user VMs are present, i.e., if Condor is allowed exclusive access to its own hardware for executing jobs. Enabling backfill and introducing user VMs causes an increase in the Condor first queued time because there are fewer backfill VMs processing Condor jobs since on-demand user VMs are also running. When backfill is enabled there is a noticeable increase in the amount of time that Condor jobs are delayed until they finally begin executing before successful completion, as seen by the numerous spikes for individual Condor jobs in Figure 8 (of which there are a total of 48). These 48 jobs actually first begin executing much earlier, as seen by the absence of spikes in Figure 7. These jobs are delayed because of the arrival of the on-demand VMs, which cause the termination of backfill VMs, preempting the running Condor jobs. Of the 48 jobs that are preempted the average amount of additional time these jobs are delayed (before they begin executing for the final time) is 627 seconds with a standard deviation of 76.78 seconds; the minimum amount of extra time that a job is delayed is 273 seconds and the maximum is 714 seconds. The 48 preempted jobs spent a total of 22,716 CPU seconds processing the Condor workload before they were preempted. The entire Condor workload required a total of 355,245 CPU seconds. Thus, for our experimental traces, the use of a backfill-enabled IaaS cloud resulted in an additional 6.39% of overhead for the Condor workload. Figure 9 demonstrates the impact that backfill has on ondemand user requests. When backfill is disabled all on-demand user requests are handled in 2 seconds or less. However, when backfill is enabled the amount of time to respond to an ondemand user request can be as high as 13 seconds, though the majority more closely match the case where backfill is disabled. The large delay in response time is when the Nimbus service must terminate (via clean shutdown) backfill VMs in order to service the on-demand user request. Additionally, because the evaluation environment consists of 8-core nodes with backfill VMs consuming all 8 cores, whenever a backfill VM is terminated to free space for an on-demand user VM (even if the on-demand user request is only for a single core), the remaining cores on the VMM node remain idle and freely available for future on-demand user VMs.

While this evaluation is based on two real workload traces, one can imagine that under some of the possible workloads, backfill VMs may be more or less beneficial to IaaS cloud providers and HTC users. Certain workloads, environment characteristics, and backfill termination policies will undoubtedly lend themselves as more beneficial to one community over the other. This is something we will consider in future work. However, our backfill solution and evaluation demonstrates that when considering a realistic on-demand user workload trace and a realistic Condor workload trace, a shared communication between IaaS cloud providers and an HTC job management system can be highly beneficial to both IaaS cloud provider and HTC users by increasing the exploitation of the cloud communication (thereby decreasing the overall cost) and contributing cycles that would otherwise be idle to processing HTC jobs

V. Related Work

Although our work utilizes backfill to achieve high exploitation of an IaaS communication, it is different from work that uses backfill scheduling to increase the exploitation of large supercomputers . Scheduling on supercomputers does not typically assume that backfill jobs will be preempted by an ondemand
request, seeking to immediately access the resources, while our work assumes this to be the default case. Instead, these backfill scheduling algorithms only attempt to backfill unused resources with requests that match the available slots both in their resource needs as well as their expected runtime. There are, however, preemption based backfill solutions that share many similar characteristics to our work. The major exception is their focus on queue-based supercomputers and our focus on IaaS cloud communications. Volunteer computing systems, such as BOINC , harvest cycles from idle systems distributed across the Internet. Major examples of volunteer applications include SETI@Home and Folding Home . These applications are designed to accommodate interruptions in service since widely distributed computers, operated by a seemingly infinite number of disparate users, cannot provide any guarantee of service. In the case of volunteer computing systems interruptions in service are usually the result of users returning to their systems to do work, systems crashing, or systems becoming disconnected from the Internet. Much research on volunteer computing focuses on the usefulness, efficiency, and failure prediction of these volatile environments . Our work focuses on providing cycles within an IaaS communication that would have otherwise been idle to other processes, such as HTC or volunteer computing, where the services may be interrupted by the arrival of requests for on-demand VMs. Applications that leverage volunteer computing systems would be ideal candidates for backfill VMs because of their ability to handle unexpected failures in service. we also leverage recovery techniques, specifically suspending and resuming VMs, to achieve high exploitation of IaaS cloud communications. While the goal of maintaining high exploitation via introducing different types of leases is the same as the work described here, the leases themselves as well as the recovery technique used, specifically that of suspending and resuming VMs, is different from the focus in our work. Instead of using suspend/resume to support advanced reservations we leverage a recovery system that uses resubmission (Condor) to ensure that high exploitation is achieved and no work is lost.

Another area that shares related themes to our work is spot pricing, as exemplified by Amazon . With spot pricing users place bids for instances and the cloud provider periodically adjusts the price of spot instances, terminating the spot instances with bids that fall below the new spot price and launching instances that meet or exceed the spot price. Our work uses the current demand for on-demand user VMs to determine the availability for backfill VMs while Amazon bases availability of spot instances on a spot price.

VI. Future Work

The backfill implementation used in this paper was an initial prototype created to demonstrate of the usefulness of combining IaaS cloud communication resources with other purposes, such as HTC, through backfill VMs. The prototype implementation used in this work is publicly available on GitHub . The 2.7 release of the Nimbus toolkit includes the official release of the backfill implementation. In the 2.7 release backfill instances are essentially zero-cost spot instances that have a lower priority than on-demand instances and spot instances. Therefore, backfill instances are preemptible by both on-demand requests and spot requests. The future work opens up the opportunity to explore different variants of the policies described in Section II. For instance, exploring finer granularity with which to deploy VMs, optimizing the backfill image deployment method, as well as termination policies. Another possible area for future work is suspending backfill VMs instead of terminating them. Such a solution may be ideal for a backfill application that does not leverage resubmission as its recovery mechanism. Another set of challenges arises if we broaden the definition of the perceptible lease, e.g., by removing the assumption that only one type of backfill VMs may be used or that only the administrator can configure backfill VMs. One simple refinement would be for the administrator to define multiple backfill VMs and have policies on how backfill resources are shared among them (e.g., what percentage of available cycles should be devoted to each). However, if users are to submit backfill VMs (i.e., the perceptible lease as defined in this paper would no longer be “fixed”) some arbitration mechanism needs to be defined for deciding between various user/instance requests. For example, AWS uses auctions to make such decisions (i.e., spot instances) but many other mechanisms could also be explored. Additionally, we could also consider different types of leases, e.g., to provide for the impact of backfill VMs on parallel jobs where all processes for a single parallel job must be available. Another set of challenges arises out of exploring various aspects of resource exploitation, energy savings, cost and pricing. An assumption throughout this paper has been that civilizing exploitation is advantageous because it leads to better resource amortization and thus lower costs per computation cycle. This need not necessarily be so: green computing techniques allowing

providers to power down a proportion of resources may be a better option in some cases, and prices obtained by auction need not necessarily be sufficient to amortize cost.A more thorough model taking into accounts these factors would be needed.

VII. CONCLUSIONS

In this paper we propose a cloud communication that combines on-demand allocation of resources with opportunistic provisioning of cycles from idle cloud nodes to other processes, such as HTC, by deploying backfill VMs. We extend the open source Nimbus IaaS toolkit to deploy backfill VMs on idle cloud nodes. We evaluate the backfill solution using an on-demand user workload and an HTC workload. We find backfill VMs contribute to an increase of the exploitation of the IaaS cloud communication from 37.5% to 100% during a portion of the evaluation trace but result in only 6.39% additional overhead for processing the HTC workload. Additionally, backfill VMs make available cycles that would have otherwise been idle to assist in processing HTC jobs. In particular, a Condor workload that originally completes in approximately 85 minutes on dedicated hardware is only delayed by 11 minutes and 45 seconds (completing in just under 96 minutes) when ondemand user VMs are introduced into the system.

ACKNOWLEDGMENTS

We would like to thank puram pradeep kuarm for his help and advice for writing of this paper and very thankfull to our faculty members.

REFERENCES

[1] Acharya A, Edjlali G, and Saltz J. “The Utility of Exploiting Idle Workstations for Parallel Computation,” SIGMETRICS ’97, pp. 225-34.

[2] Amazon Web Services. Amazon.com, Inc. [Online]. Retreived December 6, 2010, from: http://www.amazon.com/aws/

[3] Anderson D and Fedak G. “The Computational and Storage Potential of Volunteer Computing,” CCGRID’06, 2006, p. 73-80.

[4] Anderson DP, Cobb J, Korpela E, Lebofsky M, Werthimer D. SETI@home: An Experiment in Public-Resource Computing. Communications of the ACM, 45(11), November 2002, 56-61.

[5] Anderson, D. “BOINC: A System for Public-Resource Computing and Storage,” 5th IEEE/ACM Workshop on Grid Computing, Nov. 2004.

[6] Douglas Thain, David Cieslak, and Nitesh Chawla, "Condor Log Analyzer", http://condorlog.cse.nd.edu, 2009.

[7] Feitelson DG, Rudolph L. Parallel job scheduling: Issues and approaches. Lecture Notes in Computer Science: Job Scheduling Strategies for Parallel Processing, 949, 1995.

[8] FutureGrid. [Online]. Retreived December 6, 2010, from:http://futuregrid.org/

[9] Internet Retailer Magazine. [Online]. Retreived December 6, 2010,from: http://www.internetretailer.com/top500/list/

[10] Science Clouds. [Online]. Retreived December 6, 2010, from:http://www.scienceclouds.org/

[11] Smith, JE. and Nair, R. Virtual machines: versatile platforms for systems and processes. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2005.

[12] Snell Q, Clement M, and Jackson D. Preemption based backfill. In Feitelson, Rudolph, and Schwiegelshohn, editors, Job Scheduling Strategies for Parallel Processing, pages 24–37. Springer Verlag, 2002. Lect. Notes Comput. Sci. vol. 2537.