The Xen Roadmap

Ian Pratt, July 2006

Since around October 2005 the main focus of the Xen project has really been around stability: getting the 3.0.0 release out, and then ongoing hardening in preparation for FC5 and SLES10. We're now in the fortunate position of having really very few outstanding bugs, and now have excellent automated test infrastructure to help us maintain this level of quality.

It's now time to turn thoughts to further development and work out where we want to take the project toward the next major release, Xen 3.1 (or 4.0, whatever we end up calling it). This document lists what I believe are the priorities over the coming months, and hence what the core Xen team will be investing effort in.

1 Releases

Rather than the 'big bang' 12 month development cycle used in Xen 1.0, 2.0 and 3.0, we want to make the development toward the next major release far more incremental. We aim to continue the current practise of going through a stabilization phase and having point releases every 10-12 weeks. Just as with Linux, the point release will be maintained with bug fixes until the next point release.

One side effect of this approach is that at the time of a release not all features that are new in the code base may necessarily be stable in time for the release. We won't hold the release, but will simply disable such features or document the issues. The aim is obviously to avoid regressions in functionality that worked in earlier releases.

2 Performance, Scalability

During the Xen 3 development cycle the vast majority of effort went in to correctness rather than performance and scalability. It's pretty clear from looking at historic benchmark data that some performance bugs have crept in along the way: there really is no good reason why xen3 should perform worse than xen2 on any benchmark, but measurements indicate there are currently regressions on some benchmarks. Clearly, investigation is called for. I expect there to be quite a bit of low hanging fruit, in particular it is likely that various of the Linux kernel version upgrades have broken certain assumptions causing us to exercise 'slow paths' rather than the fast path we expect it to be using. These need to be tracked down and fixed.

We're actually quite well-armed with performance monitoring tools these days, so hopefully this shouldn't be too difficult: xen-oprofile is a sample-based profile system useful for looking at system-wide CPU consumption. The xen software performance counters are useful for tracking event occurrences in Xen (and hence spotting anomalously high counts of supposedly rare events), and can also be used to collect various histograms of scheduler and memory management data. There is also xentrace which can be enabled to collect timestamped trace records into a buffer to enable detailed event time-lines to be collected.

We expect the vast majority of Xen deployments to be on two socket server boxes, with some four socket, and we expect dual core to be commonplace. Hence, we believe that the sweet spot to optimize xen for is 1-8 CPU systems. 32 way and larger systems are currently supported and getting good performance on them is certainly desirable in the mid-term. However, we must endeavour to ensure that such optimizations do not harm smaller system performance. If necessary, compile time target selection could be used, but I hope this won't be required. One of the key features which will help both small and large system performance is support for NUMA-aware memory allocation. This is useful even on two socket AMD boxes. I expect to see the core Xen mechanisms in the tree in short order, but implementing page migration (particularly the policy for doing so) will be a longer term goal. NUMA topology-aware CPU scheduling will also need to be developed, and is discussed in a later section.

3 Guest API stability

The Xen 3 Guest API is intended to be a stable interface that will be maintained in a backward compatible fashion. Old 3.0 guests should run on a new hypervisor. The converse of running new guests on an old hypervisor has not been a commitment, but we should obviously plan to transition toward this. This becomes more important as soon as a given version of Xen starts gaining wide use in an Enterprise Linux distro (e.g. SLES10).

The xen guest API includes all the virtual IO interfaces (e.g. netfront, blkfront etc). We expect there to be some evolution in these protocols, but nothing that can't be supported in a compatible fashion: xenbus provides the necessary mechanisms for enabling feature selection, or even selection of alternate front or backend drivers.

So far, there has been no commitment to maintain a stable interface between privileged domains (dom0) and xen, or between xend and dom0. We currently expect xen, the dom0 kernel, and the xend tool stack to be a "matched set". These interfaces will continue to evolve for at least the next six months, so no stability is guaranteed, though interface breakage will be avoided where possible as it's clearly inconvenient for developers. Interface version numbers should be updated when this happens to avoid subtle incompatibilities.

The continued evolution of the privileged domain interfaces is being driven by a number of factors: changes such as the new pci device pass-through code, support for IOMMUs, integration of ia64 and ppc, more fine-grained security capability delegation. It should be a long term goal to stabilize this interface, but for the moment 2.6 Linux is in the favoured position of being the in-tree privileged domain OS, with NetBSD and Solaris playing catchup.

One of the key APIs we expect to evolve and then stabilize quickly in the next 2-3 months is the xen control API, encompassing all the various XML parameters that configure and control guest domains. The roadmap for the xen control stack is set out later in this document.

4 OS Support

The following OS kernels have been ported to the Xen3 32b guest ABI: Linux 2.6.16, 2.6.9-34.EL, 2.4.21-40.EL, 2.6.5-7.252, NetBSD3, FreeBSD7.0

and OpenSolaris10. Work is planned to update the Plan9 port to xen 3, and the API stability will hopefully encourage other OSes to be ported too, such as OpenBSD. The only x86_64 OS port is currently 2.6.16, but other ports are underway.

Since the 3.0.0 release there have been various backward compatible enhancements to the Xen guest API, such as the 'feature flags' and 'transfer page'. These mechanisms are intended to make backward compatibility easier to maintain in future, and will be used to enable features such as supporting 32b guests on a 64b hypervisor, running xen kernels on bare metal, and running xen kernels as HVM (fully virtualized) guests. We would urge maintainers of kernel ports to adopt the new feature flags and transfer page ASAP if they haven't already done so.

5 Getting xen support in kernel.org Linux

The work to get xen support into upstream kernel.org Linux is a crucial work item for the xen community. Although it costs considerable man effort right now, it should reduce the maintenance burden in the long run as hypervisor support will be a more visible consideration for Linux developers, and hopefully treated as a first class citizen.

The full xen support for Linux patch maintained in the main xen repository is really quite large. As well as supporting Linux as an optimized SMP guest, it has all the functionality necessary to run Linux as a domain 0 with access to physical hardware, support for IO virtualization backends, support for the control tools etc.

It is clear that getting such a large patch into kernel.org in one go is infeasible, so a more incremental approach has been adopted. Our initial aim is to submit patches that enable Linux to run as a simple Xen guest, without various of the more invasive paravirtualization optimizations that provide rather better virtual memory and SMP performance. A patchset has been prepared by Christian Limpach, Chris Wright and Jeremy Fitzhardinge, and is being iterated on LKML.

The path into kernel.org has been muddied by the discussion around VMware's VMI proposal. VMI proposes an abstraction layer for the hypervisor to guest API used for CPU and memory management virtualization, and doesn't attempt to address the virtual IO, hardware access, and control tool APIs that the Xen patches address. VMI is also currently only addressing 32b x86 (Xen is currently x86_64 and ia64 too).

Comparing VMI with a subset of the xen patch, there's actually quite a lot in common. The bulk of both patches refactor i386 code to provide hooks, and it is hoped that these common changes can be upstreamed quickly, reducing the size of the external patch that must be maintained. Rusty Russell is taking the lead on this. At some point there will be a discussion about what the correct hypervisor API abstraction is, and hence the extent to which paravirtualization can be exploited to improve performance. The Xen team will be arguing vigorously that a rich interface that is part of the kernel source (and hence GPL) is more easily maintainable, and enables 'deeper' paravirtualization, and hence more opportunities for performance enhancements.

It is conceivable that what ends up being merged into kernel.org may require some changes to the xen 3 guest ABI. We would certainly hope that these could be made backwards compatible with existing xen guest OSes, but we may have no choice other than to rev to a Xen 4 ABI. This would be a great shame as ABI stability is a key hypervisor requirement, but this would at least be a one off change.

6 Control tool stack

Following on from the discussions at the last Xen summit, a number of significant changes are planned for the xen control tool stack over the next few months. In the mid-term, the DMTF CIM model for VM life-cycle management from the virtualization and partitioning working group is likely to emerge as the standard for configuring and managing VMs. Many of the folks in the Xen community are already working in the DMTF in support of the CIM model, as well as Microsoft in their Carmine tools. The joint IBM/Novell/XenSource project to create CIM providers for Xen is of great strategic importance. However, management via CIM is quite heavyweight and intricate for some scenarios, so also having a simpler management API makes sense. Indeed, having such an API will be useful for building the CIM providers themselves.

XML config file and conversion tools. The first stage of xend development will be to switch to using XML for all configuration data. We are in the process of drafting a specification for an XML data-model for storing all

VM configuration data, and will be circulating this on xen-devel/xen-cim shortly. This scheme is 'inspired by' the CIM data model, but the hierarchy is somewhat flattened and simplified to reflect xen's requirements and provide an easy to navigate model. The intention is that XML config files conforming to this schema would replace the current python and SXP xend config files. Since this is a user-visible change, creation of migration tools will be required. Config files would no longer live under /etc, but would be loaded into xend when a VM is created, and then stored as plain XML files under /var.

xend VM life cycle management and storage extensions. Another addition will be implementation of some simple VM life-cycle management inside xend. The current tools already have a very limited form of this implemented by the xendomains script which will preserve VMs across host reboots using save/restore. Adding VM life-cycle management to xend means that we will store state for VMs even if they are not currently running. This will enable us to make operations like VM save/restore less dangerous from a user point of view by tracking the resources (in particular, disk images) they have reserved even when they are suspended. Further, support for simple storage management will be added to xend, enabling call-outs to create new disk images, create CoW snapshots etc. These will be implemented in scripts that provide the functionality for different storage backends e.g. LVM, file based, qcow, etc.

In previous and current Xen releases the protocol between 'xm' and 'xend' has not been documented and has undergone rapid change. Although the internal xmlib API has generally remained relatively stable this has not been a popular interface for developers building on top of Xen – most have resorted to using 'expect' scripts to drive the xm command line.

XML-RPC xen control API plus C/C++/perl/python bindings. It's now important that we define a suitable message protocol and associated API, rapidly switch xend and xm over to using it, and then provide client bindings for common languages e.g. C++/perl/python. Libvirt will hopefully help fulfil this role. Draft specifications for the message protocol are being prepared and will be circulated shortly. The scheme is xml-rpc based, using an SSL secured https transport to enable secure remote management, or unix domain sockets for lightweight local management.

The protocol will support a notion of logging in as a given user, using PAM to authenticate on the server. Since some of the RPC commands may be long running (e.g. a VM relocation), the protocol supports the notion of tasks running asynchronously in the background. The client can poll for their completion or list outstanding tasks.

In addition to securing the control protocol, we also need to secure the networking connections used for VM relocation. The intention is to split the VM relocation operation into two, issuing a relocate_receive that generates a token that then must be presented when initiating a relocate_send. Since these network connections are performance critical, it is important we retain the option to have just authentication without mandating encryption.

A further area where work is clearly required is developing a decent web GUI for Xen. There have been a number of previous attempts at this by various folk, but none have really gained traction. We would envisage that the GUI would be implemented using presentation layer code that would communicate with xend via the control protocol and generate HTML/javascript to return to the browser. Possibly picking one of the existing web UI's and trying to get it in-tree will help focus efforts.

7 Virtual Hard Disk Images

Xen 3.0 supports a wide range of storage options for guest virtual disks, including physical partitions (LUNs), LVM volumes, and 'loop' files. Use of loop files has proved popular as they support sparse allocation, and are very easy from a management/backup point of view (e.g. you can just copy the files around). However, the loop driver has some serious deficiencies, in particular it buffers writes very aggressively, which can affect guest filesystem correctness in the event of a host crash, and can even cause out-of-memory kernel crashes in domain0 under heavy write load. Further, using sparse files requires care to ensure the sparseness is preserved when copying, and there is no header in which metadata relating back to the VM can be stored.

Given the popularity of the file-backed model, providing a robust and high performance solution that supports it is highly desirable. Rather than using raw image files, it makes sense to move to a format that supports header metadata, sparse allocation independent of the underlying filesystem, and copy-on-write support. We have spent time evaluating a number of existing formats: VMware VMDK, Microsoft VHD, and QEMU QCOW. VMDK is a hodgepodge union of several different formats used by VMware, and doesn't have much to recommend it. Further, the licencing terms aren't entirely clear about its GPL compatibility. Microsoft's VHD format is pretty nice, but explicitly doesn't allow open source implementations. QEMU QCOW looks to be the best of the bunch: It's been part of the QEMU stable for over 18 months and had quite widespread use, and supports advanced options like compression and AES encryption as well as sparse allocation and copy-on-write.

The world doesn't need another virtual hard disk format, so with Fabrice Bellard's blessing (QEMU author) we're strongly advocating that the Xen project adopt it. The licence on the current QCOW implementation is BSD, which makes implementation with Xen and even 3rd party closed-source tools easy (e.g. Virtual-to-Physical transfer utilities).

Given the various problems with the 'loop' driver, this doesn't seem a good starting point for implementing qcow support. The easiest approach seems to be to build on the 'blktap' approach that is already in the tree and provides a way of implementing virtual block devices in user space. Work is well under way to implement a 'ublkback' driver that supports all of the various qemu file format plugins. A special high-performance qcow plugin is also under development, that supports better metadata caching, asynchronous IO, and allows request reordering with appropriate safety barriers to enforce correctness. It remains both forward and backward compatible with existing qcow disk images, but makes adjustments to qemu's default allocation policy when creating new disks such as to optimize performance.

8 Resource Control

One of the big deficiencies with Xen 3.0.2 is the static assignment of VCPUs to single physical CPUs. Although domain creation attempts to do some crude load-balancing placement on physical CPUs the assignment is not updated, which can lead to gross imbalance between CPUs if e.g. all even numbered VMs were to exit. Currently manual intervention with 'xm vcpu-pin' is required to rebalance things.

A new CPU scheduler has recently been completed which should solve these issues. It supports automatic migration of VCPUs between their allowable CPU set, and actively balances VCPUs across CPUs in an attempt to maximise throughput.

The scheduler supports a notion of 'weighted fair share', enabling the relative weights of guests to be set e.g. this guest should be able to get twice as much

CPU as this guest (assuming they are both CPU bound and runnable). Further percentage CPU ceilings may be set to constrain the consumption of a guest, e.g. limiting it to 10% of a CPU even if the CPU is otherwise idle. [This non-work conserving option is useful in hosting environments where it is sometimes desirable to stop customers having the opportunity to get used to more resource than they're paying for.]

For multi VCPU guests the scheduler tries to ensure that each (runnable) VCPU accrues CPU time in roughly equal fashion. This avoids bad interactions with the guest OSes internal scheduler. Currently, no attempt is made to gang schedule VCPUs belonging to the same guest. For most workloads this seems to work OK. In future we may have to investigate schemes that switch to gang scheduling under certain circumstances, or otherwise use bad pre-emption avoidance (e.g. don't pre-empt while kernel locks held) or bad pre-emption mitigation strategies (e.g. directed yield). It may be possible to dynamically spot groups of VCPUs that are actively communicating and gang schedule them. We will need to extend the scheduler to better understand CPU topology and make informed scheduling decisions in the presence of hyperthreading or NUMA architectures (in the short term, we can use CPU affinity settings to assist this). Due to potential information leakage through cache timing attacks between domains sharing the same hyperthreaded CPU core, we may need to ensure that hyperthreads belonging to the same core are gang scheduled for the same domain (of shared with a trusted IO domain).

One other scheduling feature that has been discussed is instrumenting domain 0 to enable the work that it performs on behalf of other guests to be accounted to those guests. The simplest way to do this is just to do some simple counting of grant transfer and grant map events for each domain, and then bill the CPU time consumed by domain 0 in proportion (with an adjustment factor to account for the CPU differences between disk and network IO). Although simple, this scheme would probably yield most of the benefit, and avoid CPU-bound domains being penalized in the presence of bulk IO.

Support for network QoS is in pretty good shape: The netback driver can implement token bucket rate limiting ("x KB every y microseconds)") on any virtual interface to enforce simple max rate control. More complex queueing and scheduling strategies can be implemented simply by invoking Linux's existing iptables and traffic_control facilities.

Some control over disk IO bandwidth is now possible using Linux's inbuilt

CFQ IO scheduler. Using ‘ionice’ it is possible to set the relative priorities for disk accesses. More experimentation is required to determine whether this will prove to be a sufficient means of control, or whether some further controls (e.g. implemented in blkback) will be necessary to provide xen admins with the features the require. Disk scheduling is a well-known thorny research problem, and best driven by user requirements.

9 HVM (fully virtualized) Guests

As CPUs with Intel VT and AMD-V support come to market in volume, “HVM” guest support is now a critically important core feature of Xen. The current support we have today is “OK”, but we know we can do a lot better, and really make the new hardware fly. This involves substantial surgery to a number of key subsystems, but we seem to be making good progress.

9.1 Shadow Pagetables

One of the most important and complex subsystems in Xen is the shadow pagetable code. For paravirtualized guests, this is typically only used when guests are undergoing live relocation, but for HVM guests it is turned on the whole time and is critical to performance. The code has to support a number of different modes of operation, and deal with the differences between guest and host page table levels – the code supports 2-on-2/3/4, 3-on-3/4, and 4-on-4.

The current shadow pagetable implementation is large and very complex, partly as a result of having been hacked on by a lot of different people each addressing their own requirements. The algorithm itself isn’t too bad, but we’ve now accumulated a lot of profile data from different OSes to enable us to do a better job this time round. We have designed a new algorithm and have embarked on a complete rewrite of the code, which will take some time complete. Because of the shadow pagetable code’s importance, testing it is a major task as the test matrix of different OS versions and configurations is vast. We will need to run the two code bases in parallel for some period of time, either in different trees, or possibly in a single tree with a boot option. We have a list of optimizations and heuristics we intend to add to the new implementation once the core is stable. In the meantime, there are various folks continuing to fix and optimize the code currently in the xen tree, which further helps inform the new design.

9.2 QEMU

QEMU has proved to be very helpful for providing Xen HVM guests with emulated IO devices. However, Xen's current "qemu-dm" code has diverged quite heavily from mainline qemu, which means that we can't as easily capitalise on enhancements made to mainline qemu, and also makes it harder for us to contribute enhancements back to Fabrice. Fixing this is a priority. We are developing a re-implementation of qemu-dm that is maintained as a patch queue against an unmodified snapshot of mainline qemu, and hope to have this ready for testing soon. Like the shadow pagetable code, this is going to require extensive testing even prior to inclusion in -unstable.

Catching up with the latest version of qemu has a number of nice side effects. It gets us Anthony Liguori's improved VNC server code for the framebuffer, removing our dependency on the rather obtuse libvncserver library. Further, it gets us support for emulated USB devices, most interesting of which is a USB mouse. The USB mouse protocol supports a mode of operation whereby it provides absolute x,y co-ordinates rather relative motion events (which the OS usually turns into absolute co-ordinates using some unknown 'black box' algorithm that has mouse speed and acceleration parameters). Being able to inject absolute co-ordinates means that it is easy to arrange for our the HVM guest cursor to perfectly track the local cursor in the VNC viewer, regardless of the user's mouse settings.

Although not immediately on the horizon, there are plans to give qemu a slightly extended role in Xen. The 'V2E' research work in Cambridge has shown that it is possible to move the execution state of a guest from a Xen virtual machine into qemu and back out again. In the context of the research work the aim was to enable high-performance taint tracking to be implemented, where the guest ran at full speed as a xen guest until it accessed a tainted value at which point execution was moved on to the qemu emulator which was able to monitor execution closely. At some point later execution would be moved back to Xen. As pointed out by Leendert, a similar approach can be used for transitioning into IO emulation, cleaning up the current interface and possibly providing performance optimizations when many IO operations are performed in close proximity. This technique also solves a thorny issue with Intel VT systems, which unlike AMD-V do not provide h/w assistance for virtualizing the legacy x86 ‘real mode’. Today, we have the ‘vmxassist’ code containing a crude emulator to try and handle this case, but the code fails in the presence of certain complex 16b applications, such as MSDOS or the SuSE graphical boot loader. Having the ability to throw execution onto QEMU which has a complete real mode emulation would solve this problem.

9.3 Paravirtualization Enhancements

Although we can run completely unmodified guest OSes in HVM mode, there are significant performance advantages that can be achieved by selectively adding paravirtualization extensions to guests. The nice thing about this model is that the OS author can boot their OS unmodified, then incrementally enhance the OS to exploit Xen’s paravirtualization hypercall API to improve performance, for example, adding paravirtualized IO drivers, then paravirtualized virtual memory and CPU operations etc.

The Xen API’s hypercall transfer page assists in allowing the precise method used to make the hypercall to be abstracted from the guest (e.g. INT82 in the full-paravirtualized case, VMCALL on VT, VMMCALL on AMD-V). However, the exact method for installing the page (e.g. by writing MSRs) and passing in-memory arguments to hypercalls has yet to be finalized, though there are prototype patches in existence. Making a decision on which approach to checkin is a priority.

Typically, the paravirtualizing extension that has most impact is switching from using IO emulation to using PV drivers. This can be done by taking the core of the existing netfront/blkfront/xenbus drivers and providing appropriate wrappers to enable them to be built against a native (non Xen) Linux tree, and thus loaded as modules into the native kernel running as a HVM guest. PV drivers can similarly be prepared for other OSes.

9.4 Save/Restore/Relocation

One of the key features missing in Xen’s current support for HVM guests is save/restore. Several subsystems need to be updated to add this support. We need to exploit qemu’s ability to ‘pickle’ the IO state of a guest and be able to pass it down a file descriptor. We need to add code to Xen to enable the state of all of the high-performance emulated devices such as the PIC, APIC, IOAPIC etc, to be read out via a the get_domaininfo dom0 op. We then need a simple update to the xc_save code to work with auto-translate shadow mode guests. The restore operation requires very similar changes, enabling device state to be 'unpickled' in both Xen and qemu. Having got save/restore working, live relocation should actually be relatively straight forward, just enabling 'log-dirty' mode for auto-translate mode guests.

9.5 SMP Guests

The current HVM code only stably supports uniprocessor guests, though work is underway to robustify SMP guest support. Xen already contains APIC and IOAPIC emulation modules, but this code along with the existing shadow pagetable code lacks synchronization in certain places. The new shadow mode code should go a long way to solving this. Work is also ongoing to add ACPI BIOS support, hence allowing the emulated platform to look like a modern PC. The first aim for supporting SMP guests is correctness, and once we have that focus on performance and scalability. Getting fair performance for two and four way guests should be achievable for many workloads, but this is an area where paravirtualization really helps. Eventually, it would be good to support CPU hotplug emulation in Xen.

9.6 Mid-term goals

Looking toward the mid-term, there are a number of projects around HVM guests that would be good to see implemented.

One of the key desires is to move the qemu device emulation out of domain 0, and run it in a 'stub domain' associated with its HVM guest. A 'stub domain' is the effectively the execution context associated with a domain that a paravirtualized guest would normally use, but is currently unused for HVM guests. Belonging to the same domain, any CPU time spent executing in the stub domain is naturally accounted to the guest. Like normal paravirtualized guests, stub domains are strongly isolated from other domains. However, given the close relationship with the HVM guest, executing transitions between the two occur faster than transitions to a secondary domain. Further, belonging to the same domain means that the stub domain can easily map memory belonging to the HVM guest. These properties mean that stub domains are an ideal place to run qemu, providing improved performance, accurate resource accounting, and surer isolation.

As a user space application, Qemu can't run in the stub domain directly, but requires an operating system kernel. The neatest way of doing this would be to link qemu against 'minios', which is effectively a library operating system for just this purpose. Since minios makes use of a broad range of libc calls, it is likely that minios will take some time to reach the required level of support. In the meantime, we can just use a xen linux kernel, with a minimal config to keep the size down. Since protection between user space and the kernel is irrelevant in the context of running qemu as the sole application, we could optimize performance by running user-space at the same privilege level as the kernel, effectively turning system calls into plain jmp instructions into the kernel followed by a ret to return.

An interesting thing becomes possible once we have qemu running in stub domains and interfacing with the HVM guest via the 'V2E' approach described previously: it becomes quite easy to enable unmodified guests to be run on CPUs that don't support VT or AMD-V. Gust execution in user space would proceed in the normal xen fashion, but any transition into the kernel would result in the guest being transferred on to qemu for emulation, which would then transition back to native execution when the guest exited the kernel. Having the emulation running in a stub domain is clearly important to allow the resource spend in emulation to be correctly accounted. Performance would clearly not be as good with VT/AMD-V approach, but this does provide a good way for folks with older hardware to experience unmodified guests on Xen.

From our point of view, QEMU's biggest failing is that the devices it emulates are quite old, and lack some facilities that could potentially lead to better performance. It would be nice to have an emulation of a SCSI HBA, as most OSes typically treat these differently from an IDE device and make more use of the ability to have multiple outstanding requests, which is essential for good IO performance in a virtualized environment. BOCHS has a simple SCSI HBA emulation, and it may be possible to use this as a starting point.

Emulating a different network device would also offer benefits. A network device that supported checksum offload, jumbo frames, or TSO (Transmit Segmentation Offload) could all offer CPU savings in the guests. Since IO performance is typically dominated by the number of 'vmexit' operations required, care is required when selecting which hardware device to emulate to ensure that the corresponding driver will generate a minimal number of exits. Older hardware is sometimes better in this respect, since the driver writer knew certain io port and mmio operations were likely expensive, and thus went out of their way to avoided doing them. Since such io port and mmio operations typically generate vmexits in the Xen case, minimizing them is helpful for performance.

10 Paravirtualized Guests

As stated earlier PV guest support in Xen looks in good shape from a stability point of view, but we do need to pay some close attention to benchmarking and tuning, particularly for large SMP guests. There is undoubtedly plenty of low hanging fruit that can be addressed to improve Xen performance – it just hasn't been a priority given the state of the competition.

One area that could certainly do with some attention is the 64b Linux guest ports. The code could do with a full code review, specifically to remove some of the unnecessary divergence from the i386 xen code, and to remove some of the modifications relative to native that are now unnecessary. There are a number of opportunities for investigating optimizations too. The current code was written assuming that the "TLB flush filter" found on AMD Opteron processors would become ubiquitous. Sadly, Intel haven't adopted this, and AMD have dropped the feature from future chips. There are a number of places where TLB flushes occur in the current code where it was expected that the flush filter would actually annul them. Given that we can no longer rely on this, some modifications to the virtual memory virtualization implementation is called for to reduce the number of TLB flushes. One particular idea that has been mooted is using the global bit in PTEs to preserve both Xen and User mappings across system calls. This needs to be implemented and benchmarked. Hopefully Intel/AMD will extend the x86 architecture with an address-space tagged TLB implementation at some point, providing a clean solution.

One area that hasn't received much attention recently is the "live relocation" feature since the 32b PAE and 64b hypervisor variants were introduced. Although the current code seems to work, it's never been 'productized' on these hypervisor variants, so shouldn't be relied upon. The new shadow pagetable code being developed primarily for improved HVM guest support should also help address this concern, greatly simplifying the current code.

We also need to do some work on the xc_restore function itself, modifying it to support lazy allocation of memory to avoid the current issue whereby when relocating a guest you (temporarily) need to allocate enough memory for the maximum memory size of the guest even if its current size is much smaller due to memory ballooning. Further, we could probably improve guest down-time by a few 10's of milliseconds if we did some streamlining of the various hotplug scripts and python code that do re-plumbing of network and block devices after a live relocation.

10.1 PV Block IO

The current block device protocols and implementations are in pretty good shape. The work to add a secondary backend implementation called 'backtap' to support high-performance file-based qcow implementation has previously been discussed. This blkfront implementation uses the same guest IO API, and can be used interchangeably with the current blkback and blktap backends.

One area that does need some discussion is whether the protocol should be extended to support in-band metadata operations, for example, enabling the front end to issue something akin to ioctl on the backend. Such requests could probably be implemented fairly straight forwardly, as a special variant of a write operation that also returns data in-place. We would need to have well defined meanings and enumeration of these ioctl-like operations since there may be entirely different operating system kernels at either end of the block protocol device channel. However, for 'occasional' ioctl operations it might be best just to use the xenbus control channel between front and backend. This can also be used for passing messages the other way. In particular, we should use this protocol to handle removable media change events better than we currently do. This mechanism could also be used to provide notification of virtual disk size changes, enabling an on-line resize tool to be invoked to provide seamless growing of virtual disks.

One extension that has been requested by at least one proprietary filesystem vendor is the ability to return metadata with both read and write responses (rather than just noting completion). This meta data is interpreted by the file system and used to optimize certain journaling operations. These extensions are currently considered rather specialized an probably mandate a separate blk protocol.

Another idea that has been mooted is supporting a scsi-level device channel protocol, and hence scsifront/back drivers. This would be useful for controlling more ‘exotic’ devices that need a richer set of operations than just read/write/barrier, e.g. a tape drive. It does introduce a whole load of fairly pointless scsi command creation and parsing at either end for the normal read/write case, which may end up resulting in measurable overhead. Experimentation is required.

10.2 PV Network IO

One key area that needs some performance tuning is the paravirtualized network device driver. The current drivers use a page-granularity protection mechanism to provide good containment and thus minimal trust between the frontend and backend drivers, but this does exercise the page protection mechanisms pretty hard.

There are a number of fairly obvious extensions to the network device channel protocol that should yield useful performance improvements. Some of these could be implemented in a backward compatible fashion by using spare fields in the current protocol, but at some point we’re likely to want to define the net device channel v2 protocol. This is not really a big deal as the xenbus control plane is there to ensure that we get the right drivers bound at each end of a device channel.

One cleanup that is required is to modify the way that the checksum offload protocol is implemented, allowing the offset of the checksum in the packet to be specified. This should clean up some of the current fragility we have in domain 0 when checksum offload is enabled and certain less common protocols are used.

Another useful improvement to the current code would be to add a flag to the data area of skb’s that have been the subject of decrypt-in-place operations (e.g. for IPSEC VPN’s), and only selectively scrub the flagged pages when adding buffers to the free queue.

The current driver uses page flipping as the only mechanism for transferring data over the device channel. The device channel mechanism is actually quite flexible, and with a simple bit of refactoring, it should be possible to give the hypervisor the option of either copying the packet or flipping the page, making the decision based on the operation size. The copy operation may become particularly attractive if high-performance multi-threaded copy engines start becoming integrated in to northbridges, as has been speculated in some quarters.

Another possibility worth investigating is the use of a semi-static shared memory buffer between the front and backend i.e. the grant table mappings are set up in advance for the buffer and left in place rather than being cycled. On the receive path, netback could directly copy packet payloads into the shared buffer from the hardware receive buffer, and then netfront could directly copy out of the shared buffer and into a local skb (2 copies, but no hypercalls).

It would be nice to eliminate the copy in netfront, but this is quite hard as the skb could end up queued for an arbitrary amount of time in an application socket buffer, hence consuming resource in the shared buffer. We could grow the shared buffer if we run out of space, but there's no real bound to the size that would be needed and the number of in-flight buffers we'd have to track. Possibly some adaptive scheme that uses copying for buffers that are likely to be long lived or when we are low on shared buffer resource would be possible. However, the downside of this approach is that guests have to invest more trust in their backend domains as they retain writable mappings to the packets sitting in kernel buffers. A buggy or malicious backend could modify the contents of a packet buffer after the guest kernel had validated it, and could very likely cause it to crash.

There are also performance enhancements to be achieved through supporting some of the higher-level features provided by modern server adaptors. Jumbo frame support would clearly be useful, both in domain 0 and from guest VMs, but many Ethernet installations still use a 1500 byte MTU, so couldn't benefit. TCP Segmentation Offload (TSO) for the transmit path would be useful, whereby the frontend passes the backend a very large 'super packet' for it to segment and transmit (ideally, passing it to the NIC to do the segmentation if we can get the super packet through the bridge and routing code). Even if we end up having to do the segmentation in the backend we're still better off than we were passing multiple packets as we've reduced overhead and made it easier to batch work across the device channel.

To support TSO, we need to extend the device channel descriptor format to flag packets that should be segmented. Since super packets are unlikely to be physically contiguous, we also need to be able to support data fragments in the descriptor format. This should be a relatively simple extension, adding chaining to netring descriptor entries.

In future, we should consider how we might export byte stream (TCP) data between domains in a high performance fashion. This would be particularly useful between VMs running the same machine, or when TCP Offload Engines (TOE) in hardware NICs become commonplace. Similarly we should have an extended interface for exporting RDMA (Remote DMA) capabilities.

Examining oprofile traces of Xen systems under heavy network load it looks like the network bridge code running in domain 0 takes a surprisingly large slice of the CPU. It's quite possible that we'd do better with some cut-down streamlined bridge code that just handles the common case that typical Xen installations use it under. At the very least, we need to do some investigation into why the current bridge code shows up in profile results as high up the ranking as it currently does.

The current netfront/netback approach is geared toward having domain 0 (or another domain) acting as the switch/router for packets between other domains. There are some circumstances where it may be more efficient to create virtual 'point-to-point' links between VMs, enabling them to communicate directly without going via a software switch/router. This would be akin to connecting two netfront devices together, allowing very high performance networking between two VMs on the same machine. This could be used in applications where there is effectively a processing pipeline of data being passed between VMs. Were one of the VMs in the pipeline to be migrated to a different machine, the netfront on each end of the point-to-point link could revert back to being connected to a netback driver, fulfilling the connection via the external network, albeit at lower bandwidth.

While adding all these extensions to the protocol we should remember that these same paravirtualized drivers are used to provide paravirtualized IO within HVM guests, and ensure that they work in both scenarios.

10.3 Client Devices

Although most Xen deployments are currently targeted at server machines, improving support for desktop and laptop usage is clearly important, not least to encourage developers to run xen on their main machines. Having excellent support for client devices such as USB and the frame buffer is clearly required, along with good power management.

Way back in the days of Xen 2 we had support for passing control of individual USB devices over to a guest VM. However, although apparently stable, the implementation wasn't ideal and was never updated to Linux 2.6. The best you can do today on Xen 3 is to assign a whole USB host controller to a guest using PCI pass-through, rather than individual hub ports as before.

There have been patches floated for adding this support back to Xen 3, but there's never been quite the impetus to get consensus and get something checked in. The situation has recently become more complicated with the USB-over-IP code now appearing in Andrew Morton's Linux tree, perhaps suggesting that the design should be revisited.The TCP transport would be replaced with a reliable byte-stream device channel transport to avoid the need for network configuration. It would be good to return USB support to Xen 3 as soon as possible, as this will solve a number of requirements around audio virtualization, mouse, scanners etc.

For PV guests, Xen currently only supports a virtual serial console. If a graphical console is required the guest needs to run its own networked frame buffer console, typically Xvnc. This is less than ideal as it requires networking to be working in the guest before the console can be used. Serial console serves perfectly well for boot, though some users get confused by the differences between a serial console and a typical Linux Virtual Terminal (VT).

The best solution to this is to implement an in-kernel fbdev paravirtual frame buffer driver, which uses a shared memory device channel to make the frame buffer available to domain 0, where it can either be rendered locally, or converted to a network frame buffer protocol. This is precisely what happens with the console of HVM guests today, so will also help unify the 'look and feel' between PV and HVM guests. Having the PV framebuffer as a kernel fbdev means that it will be able to emulate a text mode and display messages from fairly early in the boot sequence, again making operation closer to native.

In a basic implementation, the framebuffer control software in domain 0 would run periodically and either copy the shared memory region to the local display window, or generate the appropriate VNC updates. Scanning the whole frame buffer is obviously inefficient, particularly as there are frequently no updates, or the updates are quite localised (e.g. a flashing cursor). The current HVM frame buffer code implements a scheme whereby page-level write-protection of the framebuffer is used to trap updates, and hence provide a (very) rough indication of which areas of the frame buffer need scanning for updates. Since this code primarily uses VNC as a backend, the current framebuffer contents is compared against a snapshot to enable a more compact network encoding. The PV framebuffer should ideally share much of this code. Rather than using a page-fault based approach, if the guest maps the framebuffer once from a single set of pagetable pages, it may be possible to get better performance using 'dirty' bits on PTEs rather than taking write faults.

Achieving decent 2D graphics performance really requires more help than can be achieved simply by monitoring page-level granularity updates. Ideally, the framebuffer backend code would be provided with accurate bounding box update rectangles, and explicit 'copy region' and 'fill region' commands (along with a separate hardware cursor rather than just rendering it into the framebuffer). In the HVM guest case, this could be achieved by emulating a graphics card which supported (and whose common OS drivers supported) copy and fill operations. In the PV guest case, the kernel fbdev interface is not rich enough to supply this data, so this implies that we will need to write an Xserver driver module that supplies this data to the backend via a side-band shared memory interface. Requiring modification of the Xserver to achieve decent graphics performance is unfortunate, but there is precedent – this is the approach VMware have taken.

Anthony Liguori and Markus Armbruster are working on a PV framebuffer patch that addresses many of these issues and we hope to commit it shortly.

Achieving decent 3D graphics virtualization requires a substantially higher-level interface than that for 2D. With both the Xserver and Microsoft window systems moving in the direction of using 3D rendering even for desktop graphics, 3D graphics is becoming mainstream and not just the preserve of games and CAD/CAM. We will need to investigate drivers that encapsulate and transport OpenGL and/or Direct3D commands into backend domains where they can be rendered by the 3D graphics hardware. There are already a couple of projects that are investigating this is the context of OpenGL: Jacob Gorm Hansen's work presented at the last Xen summit, and a new project at CMU/Toronto based on Chromium.

10.4 Smart IO devices

There is no denying that IO virtualization in software incurs extra latency and CPU overhead relative to native. For applications that require the best possible performance, some help from the IO hardware is required. For over a decade their have been specialist 'smart' network interfaces available that have had the necessary hardware support to be accessed directly from user-level applications in a safe and protected fashion. Most of these interfaces have been targeted at the High Performance Computing market, designed for doing low-latency message passing, but some newer interfaces also support standard TCP/IP/Ethernet protocols. The requirements made of a network device to be able to provide safe direct access from guest virtual machines is very similar to those for providing direct access to user-level applications in a traditional non-virtualized environment. We anticipate many of these smart network interfaces becoming quite mainstream over the next couple of years as virtualization becomes ubiquitous. The same principles can also be applied to storage access, but in our experience the benefits are less pronounced.

Work by IBM has already demonstrated direct VM access to Infiniband hardware on xen, and the code for this is available in the ext/xen-smartio.hg tree on xenbits. In future, we expect to see support added for the Level5 and NetXen (previously known as UNM) smart NICs which are more like traditional Ethernet interfaces.

Typically, the main device driver is run in domain 0, which is responsible for allocating the NICs resources to other VMs and hence allowing them to map page-aligned control regions of the PCI device's address space into their own virtual address space. The main driver also controls what memory pages each of the virtual NICs is allowed to issue DMA operations to/from. Guest VMs use a small unprivileged driver to operate the virtual NIC's free/receive/transmit queues via their control area. The NIC typically also supports some QoS traffic shaping of outbound traffic from a virtual NIC, and may support more sophisticated receive demultiplex and inbound/outbound firewalling options beyond simple layer-2 demultiplexing. Having such features implemented in hardware is important as users typically don't want to give this up when switching from the current s/w solution.

Before the code can be included in the main xen tree we need to have decided the interface through which these smart NICs interact with Xen's memory management to ensure that a guest requesting DMAs to a page is actually the owner of that page. Since most OSes tend to re-cycle network buffers it is usually the case that network packets are received into and sent from a relatively small and static pool of memory. This is quite a simple scenario that can be dealt with by pre-pinning the buffers with the NIC and with Xen. Storage poses more of a challenge as the pages used for DMA will be spread throughout the system, particularly for writes (though writes are less latency sensitive). Clearly, making decisions on this interface should be strongly influenced by the support planned for chipset IOMMUs.

Another area where care needs to be taken is consideration of the interaction between VM relocation and smart NICs. In general, it is not safe to save/restore/relocate a VM that has direct access to hardware. Given that access to the interface is under the control of the domain 0 main driver, it is typically possible to arrange for this operation to be safe. However, relocating a VM to another physical machine would require that the destination machine had the same smart NIC hardware, which is inconvenient. We are considering the possibility of having a pluggable driver architecture that could enable the domain builder to 'slide in' a different low-level driver suitable for the new machine.

10.5 PV Filesystem-level Virtualization

Block-level IO virtualization provides an abstraction for accessing storage that users will be very familiar with. However, a filesystem-level abstraction offers a number of interesting possibilities too. This would work rather like a network file system such as NFS or OpenAFS, but without the need for networking as a shared memory device channel transport between fs-front/fsback would employed. Thus, "xenfs" would enable one domain to export a subtree of its file system to be imported and mounted by another virtual machine. A range of different semantics could be implemented, including a typical coherent shared file-space, or copy on write.

The xenfs approach is quite interesting as it provides a very high performance mechanism for guests to share data in a coherent fashion via shared buffer cache mappings to the same file. A shared buffer cache could also avoid some IO, and yield memory savings. Implementing this for PV guests requires use of a new type of page fault, "copy-to-write" which is different from "copy-on-write" in that the all mappings to the immutable old page must be updated to the new copy. Fortunately, an implementation of CTW faults for Linux has already been developed by the embedded systems community, who need it when mapping pages stored in Flash memory.

Mark Williamson is working on the XenFS implementation.

11 Core hypervisor

One of the key features we'd like to add to Xen soon is the ability to run a mix of 32b and 64b paravirtualized guests on a 64b hypervisor (just like you can with HVM guests already today).

Because of the similarity in pagetable formats, it should be possible to run 32b PAE guests on a 64b hypervisor pretty straightforwardly and with good performance. We just need to provide a 'compat32' version of the hypercall table and export a 32b version of the 'm2p' table. Since the hypervisor can live outside the 32b address space of the guest we can exploit the code in the Linux port to allow a variable sized hypervisor hole to enable us set the hole size to zero, giving the guest the full address space. Supporting non-PAE guests is also possible, but would require the use of shadow pagetables to convert between the guest and host pagetable formats. Given the resulting performance hit, it's probably best to just stick to PAE guest kernels.

Work on adding NUMA support to Xen is important now that integrated memory controllers on CPUs are commonplace. The goals for this work have been discussed earlier in this document.

Right now, Xen has no support for allowing guest kernels to exploit superpage mappings to access 2/4MB machine-contiguous memory regions – we always factor such mappings into multiple potentially sparse 4KB mappings. This has the potential to impact TLB usage for some workloads [NB: it is worth noting that both shadow-mode and direct-mode pagetable implementations mean that it is typically necessary to protect kernel mappings on 4KB page granularity so it is typically not possible to allow kernel use of such mappings.]

Xen's memory allocator is already capable of managing and allocating contiguous memory chunks. However, we would need to add some guest accounting to control the number of multi-page contiguous regions a guest is allowed to claim, otherwise we have the potential for one guest to hog all the contiguous chunks to the detriment of others. Such accounting is useful today even without superpage support as privileged guests can use machine-contiguous regions for IO purposes, and it would be good to bound the number of these.

Adding superpage support for mappings of user-space memory for HVM guests is pretty straightforward, though it will always be necessary to be able to factor one of those mappings later if maintaining the invariants of the shadow pagetable algorithm requires it. Support for direct-mode paravirtualized guests should also be possible, though is made slightly more complicated by an annoying quirk of the x86 pagetable format that means that linear mappings interact poorly with superpage entries as it is not possible to generate a trap if a super page PTE is accessed as though it is an L1 entry. Places in Xen where we access the guest pagetable via a linear mapping will need to be audited. On a 64b hypervisor there should be fewer of these as using the direct 1:1 mapping is typically preferable.

Power management is also a concern for the mid-term. We can certainly put idle processes into a deeper sleep state than we currently do. Looking even further out, we could also even adapt the Xen scheduler to make CPU clock frequency scaling decisions, and distribute load across processors to minimize power requirements. Getting whole-system (as opposed to guest) suspend and hibernate support would be a useful feature for laptop users. This shouldn't actually be too hard, to the extent that Linux supports the function running native on a given system. We will need to add stubs in Xen to do the final stage power down and following resuscitation.

IOMMUs look set to become common on future x86 server platforms, which offers a number of benefits for Xen. IOMMUs are clearly useful to ensure protection when assigning an IO device to a VM: without one a malicious or buggy guest could destablize the system or read data belonging to other VMs by instructing the device to DMA to/from memory pages other than those it owns. As well as providing protection, most IOMMUs also provide translation of DMA addresses. This means that they can be used to enable HVM guests to be delegated direct access to a hardware device as well as PV guests. Note that save/resume/relocate operations are likely not possible on guests with direct hardware access, unless the hardware and driver has been designed for this purpose.

Most IOMMUs typically don't support the fault-and-fixup style of operation that is common with CPU memory management: Any lookup failure is difficult to recover from, and likely involves resetting IO buses and devices, and likely results in lost IO operations. Xen must be able to recover from such situations.

In the case of a HVM guest that is not actively co-operating with Xen, Xen has to maintain an IOMMU data structure containing the full set of guest physical pages the guest may wish to instruct the IO device to access. It must ensure that each of those pages is pinned and hence it's ownership (or mmu type) can't change. Before a page can be unpinned and released (for example, by the balloon driver), xen must remove it from the IOMMU pagetable structure, issue a flush or invalidate, and wait for confirmation the flush has completed. On a processor supporting nested pagetables, it may be possible to share the guest-physical to machine frame translation pagetable with the IOMMU.

For PV guests (or HVM guests with PV extensions), the guest can be more actively involved with co-ordinating what pages are accessible via the IOMMU. The simplest mechanism for doing this would be to use the hooks provided by the kernel's DMA mapping/unmapping API and have these operations call down into Xen to update the IOMMU appropriately. On x86 this strategy is likely to be quite costly as there is little batching to amortize the hypercall to do the mapping, though unmapping can potentially be done lazily provided the pages remain pinned.

Driver domains need to perform IO on behalf of other domains, and the grant table mechanism gives them the ability to create temporary mappings to read/write data. To be able to deliver data in a zero copy fashion, page grants need to be extended to the IOMMU. The grant table mechanism was designed with this in mind, so it's a relatively straightforward extension. When calling into xen to map a grant handle, a guest can specify whether the page should also be added to the IOMMU. The grant unmap operation can operate in a similar fashion.

The only potential issue with this approach is that at the point that the map operation is performed the device that is going to be doing the IO may not yet be known (for example for network transmit we need to do the map operation before we can inspect the packet header and determine the destination interface). In most situations, it probably makes sense for all devices under control by the same driver domain to share the same pagetable structure, so this is not an issue. If this is not the case, mapping can be deferred to the kernel's DMA mapping functions, but there is likely to be less batching to amortize the hypercall cost.

In 32b x86 Xen, address space is at a premium. This leads to the notion of xen heap pages and separate 'domain pages'. The former may be accessed by Xen rapidly in any context, whereas domain pages must be mapped dynamically on demand. On x86_64 this distinction is redundant as Xen has a 1:1 mapping of all physical memory, and hence domain pages may be accessed efficiently. However, the current x86_64 code inherits the statically sized heap from 32b xen. As an immediate fix, this heap is arguably too small for a large x86_64 machine, and limits the number of VMs that can be started (the limit is lower than on 32b as the domain structure for VMs is larger on 64b builds). A better solution might be to unify the xen and domain heaps on 64b builds.

IBM have led the excellent work to add fine-grained access control mechanisms to Xen's low-level interfaces. However, the current dom0 control interface has a very simple 'flat' notion of privilege, and extending this to allow more flexible delegation of control over guests would certainly be desirable for some deployment scenarios.

Today, there already exists a notion of a bitmap of privilege capabilities that a guest has. (Note that this is orthogonal to the sets of physical resources it has control over such as ranges of machine address space, io ports etc). The current capability set is rather small, and could no doubt be made more fine grained.

More interestingly, it would be useful to be able to delegate privilege such as to be able to grant a domain permission to perform a certain privileged operation on some specified other domain or group of domains. This leads naturally to a hierarchical model of domain resource allocation and permission, for example allowing a domain with only a very restricted privilege capability to create a new domain by carving it out of its own resource allocation. It would then have full control over this domain, allowing it to destroy it, pause it, map its pages, attach a debugger etc.

From Xen's low-level 'datapath' point of view we want to flatten this hierarchy to keep the privilege check operations as simple as possible, with only the control operations having to worry about the extra complexity. Citing the example in the previous paragraph of having one domain build another, this should be quite achievable as some care is already taken to have the domain builder use standard unprivileged interfaces.

12 Testing and Debugging

Users require Xen to be a rock-solid stable system component, achieving greater stability than the OSes which run on it. Generally, we don't do too badly on this front. We're fortunate to have a relatively small and tight code base, coupled with significant investment in testing by a number of parties.

The tip of the -unstable tree is exercised daily by IBM, Intel and VirtualIron as well as XenSource. The XenSource test reports are viewable on the web at http://xenbits.xensource.com/xenrt, while other reports go to the xen-devel list. The continuous testing provides a good way of monitoring the progress toward stability of new features, and for flagging regressions.

In fact, most potential regressions never make it out into the -unstable tree, as they are picked up by automated testing in the staging tree, which gives changesets a grilling on three different machines (32b, PAE, and x86_64) before pushing the changesets out to -unstable. Hard-core developers wanting to see checkins as soon as they happen can do so by subscribing to xen-staging@lists.xensource.com.

Prior to the 3.0 release we released a special testing CD and encouraged users to download and run it on their hardware then upload the results. The CD provided booted a native Linux kernel followed by 32b, PAE and x86_64 kernels (as appropriate for the hardware), running a whole series of tests and benchmarks under each and recording the results to a log file which was then uploaded. The test CD proved very useful identifying machines Xen struggled on, and helped us get many of the issues fixed prior to release. Post release, the 3.0 demo CD has proved useful in debugging various platform issues reported by users, enabling them to collect and supply developers with data from both native and xen kernel boots.

One avenue we are investigating to help find bugs is submitting Xen to the OSS code scanning programme run by Coverity with funding from DARPA. Coverity's tools have proved useful in finding bugs in a number of other kernel-level projects, so it makes sense to investigate. The tool will likely generate a long list of possible issues, which will require a concerted effort from the community to investigate and classify, and fix the subset that are real bugs.

Even with the best possible test and QA programme, some number of in-field crashes will be sadly inevitable. In these circumstances, it's important that developers can gather as much information as possible from the incident. For user-space incidents, 'xen-bugtool' is useful for collecting log files and system details. For more serious failures of domain 0 or xen itself we can't rely on things getting logged in standard files, and need support for taking a system core dump. Horms is doing great work to get kexec working in domain 0, meaning that we will be able to use kdump to write out a system core.

Dumping a full copy of system memory can be quite time consuming on a machine with lots of RAM, so we will possibly have to consider putting a little more intelligence in the dump routines to harvest the most useful information e.g. a quick dump that concentrates on CPU register state and stack information for xen and dom0, and a more comprehensive dump that collects all xen and dom0 state and just register and stack info for other guests. We certainly need tools to help pick apart these dumps and turn them into a form that gdb can load to assist examination.

For xen developers there are a number of debugging aids that can called on. There is a serial gdb stub that can be connected to remotely and used to examine both xen and domain 0. Since this halts the system while the debugger is connected, it typically can't be used on production systems. One frequently useful tool is the debug console, which by default is accessed by hitting ctrl-A three times on the serial console. Hitting 'h' gives a help menu of features the console can perform. It's main use is for diagnosing the state of the system in event of a hang. Register dumps can be used to see what all the CPUs are doing, whether guests are servicing interrupts etc. The console also provides access to the software performance counters and other statistics.

Debugging guest VMs can be achieved using Xen's gdbserver support. When gdbserver is started in dom0 against a particular VM, it effectively implements gdb serial stub functionality on behalf of the guest, enabling gdb to connect to the gdbserver via a TCP port. gdbserver has recently been extended to support both PV and HVM guests. If getting register and stack state for all the VCPUs from a guest is all that's required, the 'xenctx' command is quicker than firing up gdb.

When guest VMs crash we currently have the option to preserve the VM, enabling debugging via gdb or xenctx. However, it would be useful to have the option to write out a core image of the guest VM. This is not quite the same as doing a VM save-to-disk operation, as we would like to write the image out in a form that gdb could then inspect. The old xen 1.2 tools had basic guest core dump support, and it would be good to see this feature back in Xen 3.

13 Misc

We need to add a hypercall to xen to allow a guest to extend the size of it's grant table. This should be straightforward, but in its current absence is the restriction which leads to the current "max 3 VIF's per guest" limit.

Actually, this restriction could also be solved by making the allocation of grant table entries to a VIF dynamic, as already happens with the blkfront driver. We should implement both strategies forthwith.

It would be useful to add PV extensions to guests to assist in taking consistent filesystem snapshots, for example, when creating “template VMs”. The obvious way of doing this would be to extend the xenbus mechanism used to deliver shutdown requests and ‘magic sysrq’ operations. Adding the ability to issue “sync disks and pause” and “remount filesystems read-only and pause” would be useful.

We need to make a few simple extensions to the control API to allow the set of CPU feature flags exposed to a guest VM (both HVM and PV) to be “cooked” such that features may be hidden. This is useful in a heterogenous CPU environment where it may be desirable to only allow guests to see a lowest common denominator set of flags such that save/restore/relocate images remain portable.

The linux guest code currently implements a strict priority ordering over how it services pending event channels. We may wish to replace this with a scheme that uses round-robin servicing of events within each group of 32 event channels, thus providing a hybrid approach that supports both prioritization and fairness. This is purely a guest issue, and the current strict priority scheme doesn’t seem to cause any problems today.

Currently, each PV guest is allowed just a single virtual serial console device. Although once we have PV framebuffer support (and associated virtual terminals) virtual serial console support will be less important, it would still be nice to support multiple consoles per guest. This should be relatively straight forward, requiring updates to the console frontend driver, xenconsoled, and the tools.

When using ‘xm mem-set’ commands to control the amount of memory in a guest its currently quite easy to set the target too low and create a ‘memory crunch’ that causes a linux guest kernel to run the infamous ‘oomkiller’ and hence render the system unstable. It would be far better if the interaction between the balloon driver and linux’s memory manager was more forgiving, hence causing the balloon driver to ‘back off’, or ask for more memory back from xen to alleviate the pressure (up to the current ‘mem-max’ limit). The hard part here is deciding what in the memory management system to trigger off – at the point where the oom killer runs the system is typically already unusable, so we want to be able to get in there earlier. Seeking advice from Linux mm gurus would be helpful.

When PV guests boot, the kernel and initial ram disk images must be supplied to the domain builder. This is not unlike what happens when a physical machine boots using PXE. However, from a maintenance point of view it is very convenient to store the kernel and initrd in the guest virtual disk image, where it can be kept in-sync with its kernel modules. This has led to a couple of different schemes to read the kernel and initrd from out of the guest filesystem as part of the PV guest domain building process. pygrub uses libext2/libreiser to read the images out of the guest filesystem, whereas domUloader mounts the guest filesystem in dom0 to extract the files. The latter suffers from potential security issues in the presence of maliciously crafted filesystem images, but is otherwise simpler to set up.

A better solution to this problem would be to use a bootloader run within the guest domain. This could be done using a cut-down linux instance that kexec's the final guest kernel, or by porting a bootloader such as grub to be able to use the PV net/block devices. Grub2 has quite wide filesystem support and is considerably easier to work on than the old grub code, so adding these drivers may not be that difficult. Grub supports a wide range of filesystems, including UFS that used by Solaris. [NB: is UFS supported in Grub2?]

The save/restore/relocate in Xen provides almost all we need to be able to take copy-on-write snapshots of VM's memory, to be used for rollback or to checkpoint long running jobs (if the guest is communicating with other machines then the wider effect of such a rollback must be considered). To be able to support checkpoints, we need to extend the tools to coordinate snapshotting of virtual disks with taking the execution state snapshot. As well as enabling such checkpoints to be initiated from the control tools, perhaps providing the ability to trigger them from within the guest would be useful too. A natural progression from supporting checkpointing would be to enable "VM forking". Rather than creating a read-only checkpoint, the VM effectively becomes cloned, running in a different domain, writing to a CoW snapshot of the disk. For VMs that have network access a mechanism is needed for enabling the IP configuration of the guest to be updated for the cloned VM. It should be possible to add code in the PV 'resume' path to change the IP address and update listening sockets while closing open connections.

Xen currently lacks support for devices that require ISA DMA (DMA below 16MB). Although ISA devices are rare these days, some PCMCIA cards have the same restriction (particularly older WLAN cards), creating problems for some laptops. We need to assess whether adding such support is going to be worth the effort or not.

Adding support for call graph support into xen oprofile would be useful. This would involve capturing the first few items on the stack as well the EIP (though there would be issues when frame pointers are missing).

Hardware performance counters are currently considered to be system-wide, which works great for xen oprofile, but prevents guests from profiling themselves. Adding support to allow performance counters to be efficiently contexted switched between domains would be useful (though use in this manner would be mutually exclusive with system-wide use).

Although efforts are made to ensure the security of the xen hypercall API, it wouldn't hurt to do a complete code audit: it is critical that unprivilged guests should not be able to access data that doesn't belong to them, or crash the system. Efforts should be made to bound the scope of 'denial of quality of service' attacks too.

A useful tool to help test the integrity of the xen guest API would be a xen equivalent of the 'crashme' tool used for testing the Linux process API. The tool would attempt to upset Xen by operating like a normal guest but using random numbers to perturb hypercall arguments, make 'difficult' corruptions to pagetables and other structures etc. (e.g. mutually recusrsive pagetables would be ain interesting one).

Finally, its about time we did another iteration on the user manual, wiki and other documentation to ensure its up to date and relevant.

14 The IA64 ports

The IA64 port is making excellent progress with contributions from HP, Intel, Fujitsu, Bull and VALinux to name but a few. Base performance is currently excellent, incurring an typical overhead of around 2% on native. Multiple Linux domains are supported (including support for virtual SMP), and clean shutdown has been completed. In addition the source code has been reorganized to produce a clean abstraction layer at the source level; with this patch applied, the resulting kernel binary can be run both under Xen and on bare metal.

Xen/ia64 is also getting close to feature parity with the x86 ports. Virtual block device using the standard backend/frontend model has been integrated, and the same set of control tools and commands are used to manage the system.

There is also support for VT-i — the hardware virtualization e-technology for IPF — which allows the running of completely unmodified guest operating systems along-side paravirtualized domains. Once again, the control tool set has been aligned with the VT-x model to maximize compatibility.

In terms of hardware compatibility, it has been extensively tested to operate on the following configurations:

- Intel Tiger4
- HP rx2600/rx1600/rx2620 (and likely any HP zx1 based system)
- Bull NovaScale 4000 and 6000 systems, with 5000 to follow
- Fujitsu Primequest series

In terms of future work, the main areas are:

1. Completing/stabilizing the work to give each guest a virtual physical address space;
2. Implementing save/restore and migration;
3. Providing support for driver domains; and
4. Additional stability / performance testing and improvements.

15 The PowerPC Port

The PowerPC work, led by IBM, is focusing on the PowerPC 970 processor which includes hardware extensions designed to support paravirtualized operating systems. An additional parallel development task is looking at getting Xen running on 970s with hypervisor mode disabled (e.g. Apple G5 systems).

The port doesn't use the standard split driver model yet, but is running guests. We hope that the code can get merged into -unstable in the next couple of months; various changes to support this forthcoming merge have already been incorporated (e.g. the use of `copy_from_guest()` which on

PowerPC uses physically addressed scatter/gather lists). Further changes to common code will also be required, which may modify the dom0 API, and so we need to ensure that people have advance warning before any such changes go in.

Power Xen currently only runs on the Maple 970 processor evaluation board, but support for IBM JS21 blades and other 970-based machines will follow soon.

16 Aims for next two releases

16.1 3.0.3 aims; end July to synchronize with FC6 freeze

- new CPU scheduler
- cow file-backed virtual hard disk support
- qemu-dm updated to latest qemu version, stored as patch queue
- basic NUMA memory allocator support
- kexec support
- xend VM life-cycle management
- new shadow pagetable code?

16.2 3.0.4 aims; end Sept

- new xml-rpc control API
- simple storage management in xend
- qemu 'v2e' integration
- PV network virtualization improvements
- PPC merge?
- PV USB support ?

17 Task priority list

Schedule:
1 - 3.0.3: end July
2 - 3.0.4: end Q3 2006
3 - Q4 2006/Q1 2007
4 - beyond

These priority ratings indicate where various features could be targeted to land on the roadmap. It reflects where the xen core team is planning on concentrating its effort, coupled with input from work that we already know is happening in the community. Obviously other folk will have different ideas of priority and will scratch their own itch and submit patches. We should obviously try to iterate in order to get such patches included ASAP, regardless of the below suggested schedule.

Sched	Area	Description
1	tools	xend VM life-cycle management
1/2	tools	XML config file and conversion tools
1/2	tools	standardized xen control API: xml-rpc over https/unixdomain sockets
1/2	tools	C++/perl/python bindings for control API
2	tools	simple storage management in xend
3	tools	revive guest coredump support
3	tools	split VM relocation operation into two parts and authenticate
3	tools	DMTF CIM providers
3/4	tools	Web GUI for Xen
1	storage	blktap (or other) support for file-based virtual disk storage.
2	storage	blktap plugins for common formats
2	storage	optimized qcow implementation
3	storage	consider adding write accounting/throttling on current loop driver
3	storage	support for block IO QoS. Use CFQ and ionice, or implement in blkback?
3/4	storage	'ioctl' support between blkfront/back
3/4	storage	media change, size change event propagation to guest userspace
4	storage	consider SCSI level storage virtualization option
1/2	network	TCP Segmentation Offload support in device channel
2	network	checksum offload cleanup
2	network	hypervisor chooses to copy vs. page flip
2	network	dynamic allocation of grant table entries; grant table resize
2	network	investigate whether bridge code needs to be 'streamlined'
2	network	jumbo frames support in dom0 and device channel
3/4	network	investigate static shared buffer approach
3/4	network	TCP Offload Engine support in device channel
3/4	network	investigate high-performance point-to-point link support
4	network	RDMA support in device channel
1	xen	extensive benchmarking and perf tuning
1	xen	CPU scheduler that balances VCPUs, implements weight & caps
1/2	xen	initial NUMA mechanism checkin
2	xen	live relocation tuning, robustification, tools safety interlock
2/3	xen	support for running 32b PAE guests on a 64b hypervisor
2/3	xen	improved NUMA policy code
2/3	xen	add order>0 guest memory allocation accounting
2/3	xen	extend x86_64 heap size; merge xen and domain pools
3	xen	investigate bad pre-emption avoidance/mitigation strategies
3	xen	add superpage support for PV guests
3	xen	IOMMU support: isolation of devices to domains; grant table integration
3/4	xen	lazy memory allocation for live relocation of ballooned guests
4	xen	fine-grained delegation for dom0ops; hierarchical resource model
4	xen	power management enhancements: CPU sleep, freq scaling
4	xen	power management enhancements: suspend/hibernate
4	xen	accounting and billing time IO domains spend on behalf of guests

Sched	Area	Description
1	hvm	fix current shadow pagetable code, add PAE-on-PAE mode, SMP safety
1	hvm	upgrade QEMU version, maintain as a patch queue
1/2	hvm	rewrite shadow pagetable code to optimize, simplify
1/2	hvm	finalize interface for making hypercalls from VT guests
2	hvm	HVM save/restore support; qemu, xen, and tools changes
2	hvm	basic SMP HVM guest support; ACPI tables, locking safety
2/3	hvm	change QEMU-xen interface to use the 'v2e' approach
2/3	hvm	SMP HVM guest performance and scalability
2/3	hvm	support real superpage mappings for HVM guests (after adding accounting)
2/3	hvm	implement high-performance SCSI HBA emulation
2/3	hvm	implement high-performance Ethernet emulation
3	hvm	live relocation. Add log-dirty support
3	hvm	move QEMU into a 'stub domain' linked against a linux kernel
3/4	hvm	move QEMU into a 'stub domain' linked against a minios
4	hvm	HVM hotplug CPU emulation
1	linux	extensive benchmarking and perf tuning
1/2	linux	SMP scalability improvements
1/2	linux	work to get xen in to kernel.org linux
2	linux	code review of x86_64 port
2	linux	investigate proposed x86_64 optimizations
2	linux	improve interaction between balloon driver and page allocator to avoid memory crunch
2/3	linux	support for multiple virtual serial consoles
3	linux	consider hybrid round-robin/priority scheme to service event channels
1/2	client	basic kernel fbdev paravirtual framebuffer implementation
2/3	client	USB virtualization; investigate USB-over-IP code
2/3	client	Xserver support for 'h/w cursor', copy rect and fill rect
3/4	client	OpenGL/Direct3D virtualization
2/3	misc	support for dom0 kexec/kdump to get a machine core
3	misc	infiniband direct guest IO support (finalize interface, merge xen-smartio.hg)
3	misc	support for hiding CPU feature flags from guests (PV and HVM)
3/4	misc	smart NIC direct guest IO support
3/4	misc	submit xen for scanning by Coverity tool; investigate warnings flagged
3/4	misc	tools support for doing auto CPU/memory resource allocation across VMs
3/4	misc	support to checkpoint/rollback guests
4	misc	port Grub2 bootloader to net/blockfront devices
4	misc	investigate 'pluggable driver architecture'
4	misc	xenfs filesystem-level virtualization; shared buffer cache
4	misc	do we need support for ISA/PCMCIA DMA (below 16MB)?