
Support for enterprise consolidation of I/O bound services



Åge Kvalnes^{1,2,*}, Dag Johansen^{1,2}, Pål Halvorsen^{3,4}, Carsten Griwodz^{3,4†}

¹ *Department of Computer Science, University of Tromsø, Norway*

² *Cornell University, USA*

³ *Department of Informatics, University of Oslo, Norway*

⁴ *Simula Research Laboratory, Norway*

SUMMARY

In this paper, we evaluate the performance effects of I/O bound workloads on a specific virtual machine, an important component of an enterprise cloud computing infrastructure. In particular, we demonstrate that 1) the I/O workload of one guest system may adversely affect the I/O performance of another for the XEN hypervisor, and 2) the general I/O performance is degraded due to various overheads. Next, we have devised a light-weight, complementary, backwards-compatible alternative to hypervisor-based virtualization techniques called BONSAI. Our software provides low-overhead I/O performance isolation by transparently applying traffic shaping to system calls in a cost-effective manner. Using this system, I/O resource consumption can be controlled at a very fine granularity. Furthermore, using video streaming experiments, where we limit the I/O bandwidth available to each service, we show that we are able to achieve the

*Correspondence to:

Åge Kvalnes, Department of Computer Science, University of Tromsø, N-9037 Tromsø, Norway

†E-mails: {aage, dag}@cs.uit.no, {paalh, griff}@ifi.uio.no

Contract/grant sponsor: Norwegian Research Council; contract/grant number: iAD/174867

Received date 1

required level of resource isolation on a per process basis with only a negligible CPU overhead and without reducing the I/O performance.

KEY WORDS: Service Consolidation; Hypervisor Overheads; I/O Performance Rate Limitations

1. Introduction

Emerging cloud computing technologies are primarily intended for Internet users and their need to minimize capital expenditure and complexity. The basic idea is to provide compute and storage resources as a utility, with low technical and economical barriers for remote users to enter. In such a cloud, tens, if not hundreds, of thousands of computers have to be orchestrated and managed, they have to be available 24/7, and they must be secure and reliable. Remote users can now share the cloud infrastructure, but individually, they get the illusion that the cloud infrastructure is his or her dedicated computer. Amazon Elastic Compute Cloud [30], for instance, provides on-demand allocation of resources so that remote users can deploy their own software and rent cloud resources, but without the cost and complexity of buying and managing these resources. *Resource elasticity* is important in this context, a property ensuring that resources can be rapidly allocated (or de-allocated) on demand without cloud users having to plan ahead of this provisioning.

We conjecture a second wave in cloud computing, where large corporations not willing to outsource internal applications and data are using in-house *enterprise cloud* technologies. These private clouds are not made available to the general public, but used to host their enterprise applications and services. This wide variety of applications and services, core to

their business models, must be extremely available and reliable, and they must scale rapidly on demand. This is normally supported by over-provisioning of resources and careful human management, but an elastic enterprise cloud solution could potentially utilize these commodity resources better.

Our conjecture has emerged by working closely with, in particular Schibsted[†], one of Europe's largest media houses with operations in 21 countries. Such a modern large media house currently provides a blend of Internet services that are exposed to temporal and spatial load variances. This includes multiple news-on-demand services, (until recently) a full fledged Internet search engine, a very popular e-commerce auction service, social networking services, and continuous media entertainment applications. Of special interest for us in a research context, has been the very popular IP based streaming service of Norwegian soccer in real-time, a Schibsted service competing with traditional television based broadcasting schemes.

To investigate enterprise cloud computing infrastructures properly, we decided to build a prototype of a practical next generation soccer streaming service [21]. This is a service that through a recommendation or search interface, is used to recommend, order, or produce personalized topic-based entertainment. The end result is a service that in less than a few seconds provides, for instance, a customized video produced on-the-fly and delivered Trans-Atlantic to a cellular containing "2 minutes, soccer player number 8 or 14 from favorite team, sliding tackles with penalty cards, this season". Timing plays a crucial role in this service, and

[†]<http://www.schibsted.com/>

user-perceived quality will degrade if insufficient I/O resources are provided, i.e., raising the need for effective and low overhead I/O isolation between the consolidated services.

Virtual machine technologies are a core component of any cloud infrastructure, a technology presumed to provide this type of resource isolation. We have been interested in evaluating this assumption more thoroughly. In particular, we are evaluating the XEN [4] hypervisor in this context, one of the popular virtual machine technologies today used by, for instance, Eucalyptus [29] and Amazon Elastic Compute Cloud [30]. An additional motivation for this evaluation is that several researchers recently have identified lack of I/O isolation (e.g., [33, 26]) and quantified the overhead (e.g., [27, 34]) of hypervisors.

The rest of the paper is organized as follows. In section 2, we describe one of our example application areas which is video search and streaming. In section 3, we describe some important properties of server consolidation, and section 4 evaluates XEN in the context of I/O bound workloads. Section 5 describes the implementation of BONSAI. Experiments and results are presented in section 6, and our approach is discussed in section 7. Section 8 presents related work. Finally, we summarize the paper in section 9.

2. Application Area

Entertainment and news production, dissemination, and end-user experiences are currently ripe for radical changes. Emerging Internet technologies and services like, for instance, blogs, wikies, YouTube, BitTorrent, Flickr, Twitter, and Facebook are paving ways for a new generation of multimedia consumers. The shift from traditional television broadcasting to IP based streaming of video content is yet an emerging application area in this context.

This section provides the overall ideas and motivations for investigating an alternative approach to virtual machines. For example, in addition to traditional news video streaming, Schibsted, through their portal VG Live (<http://vglive.no/>) recently started live (and stored) streaming of soccer matches in Norway. In this context, a traditional video streaming prototype and a compelling video on demand production and dissemination prototype are briefly presented to give contextual information for the I/O isolation research we have carried out. Both these kind of streaming systems are frequently used today and have the same high, timely I/O rate requirement, but they differ slightly in their way of operation. Also, the resulting overall software architecture for this class of media on demand environment is devised to illustrate where the I/O isolation research belongs in the larger context.

2.1. Komssys: Video Streaming

Komssys (KOM streaming system) is an experimental system for distributed audio/video (AV) streaming systems [41]. Initially intended as an RTSP server front-end for the IBM VideoCharger server and native plugin for JMF-based players, it has developed into a complete suite of standard-compatible RTSP/RTP server, proxy, and client. Komssys has been used for a variety of experimental investigations of protocols, adaptive streaming, server architectures, and distributed streaming architectures.

Komssys supports several encoders and packagers for diverse encoding formats, including experimental layered formats. Multiple components can be combined to form a processing pipeline using different stream handlers (SHs) [24] providing different functionality. Native

Komssys stream handlers form a zero-copy data path, using the Rope abstraction [8] for fragmentation and to add headers.

Using Komssys, video data is stored at the server in single, self-contained video files. These are streamed according to the consumption rate (usually determined by the file sink SH) using RTSP for control and RTP for data delivery as shown in figure 1. When the user has sent an RTSP PLAY message, the server starts to push out video to the client until the video ends or a new RTSP control message arrives. Inside the server, video data is read from the storage system for example by using `read()`, an RTP header is then appended by the application, and finally the RTP packet is sent using `send()` or similar calls. As shown in the figure, Komssys can also use TFRC for congestion and rate control, and in combination with layer-coded video like SVC and SPEG, it can use the congestion information to drop or add video quality layers in the encoder SH. Nevertheless, I/O is the main bottleneck, and fast access to storage and network resources is crucial for a user's experience of the streaming service.

2.2. DAVVI: Next Generation Multimedia Search Technology

For the next generation multimedia technology for entertainment production and dissemination, we have developed DAVVI, a prototype of a multimedia entertainment platform that provides brand new, personalized user experiences [21]. Through applying search and recommendation technologies, end-users can be recommended, order, and produce their own topic-based entertainment and news content by combining sequences from the whole video archive into one personalized and continuous video playout.

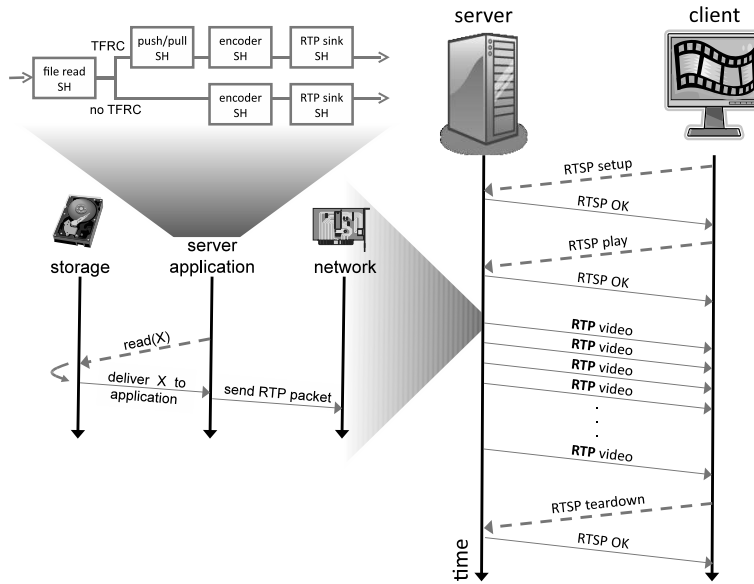


Figure 1. Packet exchange and I/O operations in Komssys.

We have been using DAVVI in practical settings with data and metadata from Schibsted's archives of Norwegian soccer. Several key software components constitute the system, as illustrated in Figure 2. To improve search accuracy and recommendation, we index the video data using metadata both from automatic analysis tools and numerous reliable related public sources of information. This includes live text commentary web pages. However, important in this context is the I/O intensive data delivery sub-system, where DAVVI uses a torrent-like data delivery approach.

First, the video is segmented into many, same length segments, which are coded as small self-contained videos to support arbitrary combinations. Furthermore, we encode each segment in multiple qualities, and thus bit rates, to support different end-user platforms and to adapt

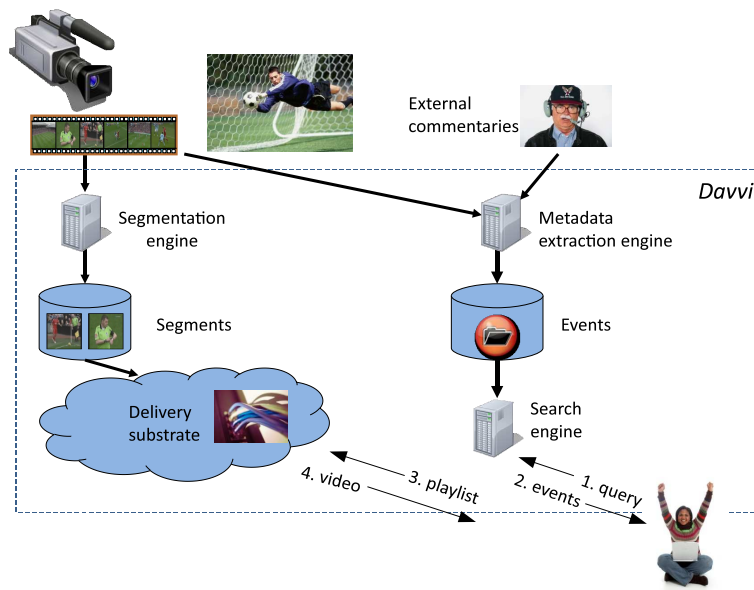


Figure 2. The DAVVI architecture.

the video quality to oscillating (network and CPU) resource availability. Next, we distribute these video segments to traditional web-servers at multiple locations to allow a torrent-like data delivery using HTTP GET requests for each segment as shown in figure 3. This resembles architectures of popular systems like, for instance, Move Networks [1] and Microsoft's Smooth Streaming [2].

At the web-server, the challenge is to support download of ten-fold thousand video segments (a traditional 90 minutes soccer match is broken down to over 2700 small, indexed videos) requested by concurrent users. Triggered by the HTTP GET, the video segments are retrieved from disk and sent over the network. In this respect, modern web-servers often use `sendfile()`

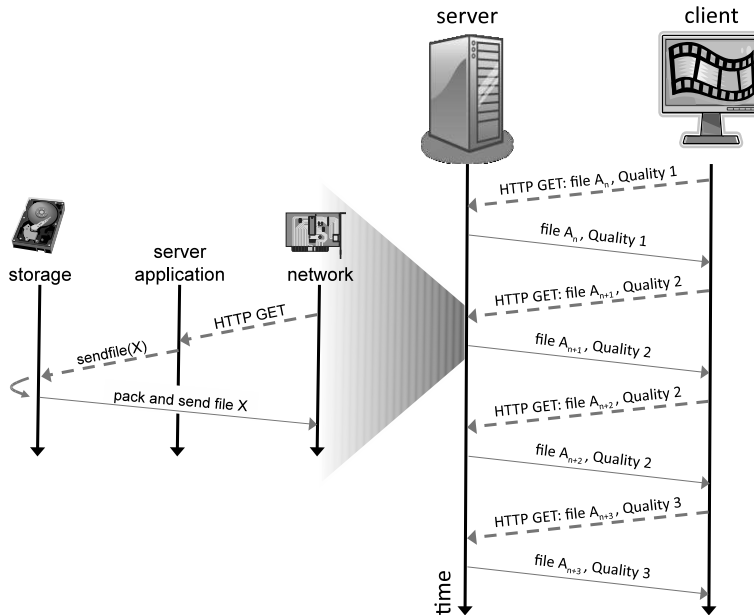


Figure 3. Packet exchange and I/O operations in DAVVI.

to avoid copy operations, but timely and predictable I/O from both the storage and the network sub-systems is a necessity to support a hiccup-free video playout.

Finally, at the client side, a specialized player and segment download proxy is used to provide seamless video playout. The download proxy monitors segment buffer level and download speed to determine video quality, and it delivers the segments in playout order to the video player.

2.3. Asymmetric Architecture Emerging

An interesting open research question we are left with after building this next generation video search and production prototype, is how the supporting infrastructure should be architected.

We have already conjectured that it will be built out of enterprise data centers with cloud computing software. Back-end servers in the enterprise cloud forms the root of a dissemination network of intermediate distributed web servers, with leaf nodes supporting the end-users. These leaf nodes also lend themselves to future peer-to-peer enhancements.

The interesting research problem for us is I/O resource elasticity, the scalability of servers performing I/O under varying load, and video playout quality as perceived by a varying number of end-users. This is also a practical and real problem in the video streaming domain. For instance, informal experience with commercially deployed IP based dissemination of soccer videos gives us a varying delay relative to actual real event time. This depends on the number of end-users simultaneously watching and their locality. We experience not only seconds delay, but sometimes closer to a minute relative to real time. This is not acceptable for the end-user in the longer run.

Throwing more resources at a scalability problem is a solution that often works. These resources are today primarily provided through virtual machines (VMs). This motivates the focus of the rest of this paper, a thorough evaluation of how well one concrete virtual machine behaves given the properties and constraints of our video streaming application area.

3. Consolidation of services

Application service providers can drive down infrastructure costs by consolidating services. One important reason is that the popularity of the various services may differ greatly depending on time of day and day of week. For instance, many use a news-on-demand service in the middle of the day [22], whereas video-on-demand services are most popular in the evening [39]. Thus,

different services are likely to experience dynamic workloads with potentially different peak hours.

A challenge is to prevent that a load peak at one service influences the performance of another. To provide acceptable services, each user should receive a minimum amount of resources, despite high load at other services. Thus, reliable access to I/O resources is needed.

One approach is to carefully attribute resource usage to the various services, and employ schedulers that make use of this resource usage information to performance isolate services from each other. When the system goes into a highly loaded state, performance isolation ensures that competing consumers are *locked out* of the performance isolated consumer's share, while the isolated consumer may be *locked into* its share and does not compete against others. While performance isolation typically guarantees full availability of a minimal amount of resources to all customers, it is not always the best solution for service providers. It is often appropriate for a service provider to over-commit his resources in such a way that minimal service is provided to customers with a given probability. Thus, the long-term resource saving gain is higher than the penalties inflicted for not satisfying the service level agreement.

Currently, service providers frequently use VMs to provide the required performance guarantees, e.g., Eucalyptus and Amazon Elastic Compute Cloud are using XEN [29, 30]. In terms of CPU and memory resources, a VM is an appropriate tool to achieve the necessary performance isolation. I/O isolation, however, has received little attention, even though many services require means to enforce resource availability.

4. Virtual machines

Hypervisors are a virtualization platform that allows multiple operating systems to run on a host computer at the same time, and they are becoming commonplace with VMs like XEN [4] and VMware [13]. Here, the virtualized environment between the computer platform and its operating system enhances the sometimes weak form of isolation provided by the process abstraction in commodity operating systems. VMs provide execution “sandboxes” with improved isolation properties, compared to running multiple processes on the same instance of an operating system.

Until recently, the functional aspects of VMs have been in focus. Comparatively, problems with interaction among virtualized environments hosted on the same machine have received less attention. In our work, we are studying this interaction problem. As a preliminary result, we address the issue of I/O performance isolation for I/O intensive applications that have rate or timeliness requirements. As a representative of the VM software, we have experimented with XEN, which is designed to host a moderate number of virtualized environments.

4.1. Lack of I/O isolation in XEN

Several algorithms for I/O performance virtualization exists. However, a challenge is that there might not be absolute isolation and fairness between the virtualized guests; the I/O operations of one application may influence the performance of another [33]. To evaluate the I/O isolation properties of XEN, we ran experiments on a XEN-enabled Linux 2.6.11.4 kernel with two XEN guest environments (referred to as XEN-a and XEN-b). We used the flexible I/O benchmark (fio, version 1.17.2) to read data from the same fragmented disk. Any benchmark program

producing a predictable rate could be used to generate data requests reflecting the streaming scenario we have in Komssys and DAVVI. However, we chose fio due to its fine granular, per-time interval reporting of the achieved rates, i.e., to observe if the timeliness of the data delivery in our streaming scenario is achievable or not. Furthermore, XEN-a had one reading thread trying to read a 5 GB file at a rate slightly above what we measured to be the maximum (~ 36 MBps). XEN-b started two threads after 20 and 40 seconds, both trying to read 2 GB files at 20 MBps. We performed experiments using both the NOOP[‡] and the CFQ[§] schedulers to investigate if there are differences between simple and more complex, but modern, schedulers, with respect to isolation properties.

The results are shown in Figures 4 and 5. Figure 4 shows the achieved bandwidth for each second when XEN-a is running alone. The plot shows several different runs, using both the NOOP and the CFQ scheduler. The experiment indicates a maximum bandwidth of about 36 MBps when running alone, and the minimum was measured to about 26 MBps regardless of the I/O scheduler.

When comparing this to the results of the competing scenario (shown in figures 5(a) and 5(b)), we observe that the competing XEN guests strongly influence each other. In the NOOP experiments, we notice that when XEN-b starts the two read processes after 20 and 40 seconds, the XEN-a performance immediately drops from 36 MBps to 15 MBps and 3-4 MBps,

[‡]NOOP works by placing all requests into a simple unordered FIFO queue, and implements only request merging.

[§]CFQ works by placing synchronous requests submitted by processes into a number of per-process queues and then allocating time-slices for each of the queues to access the disk.

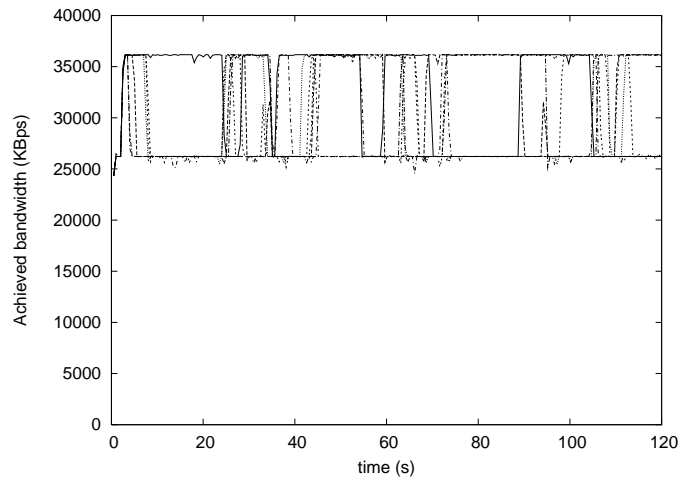
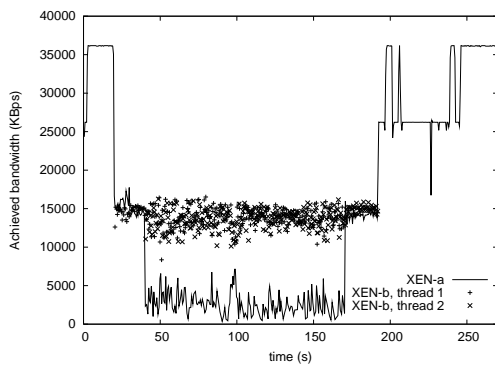
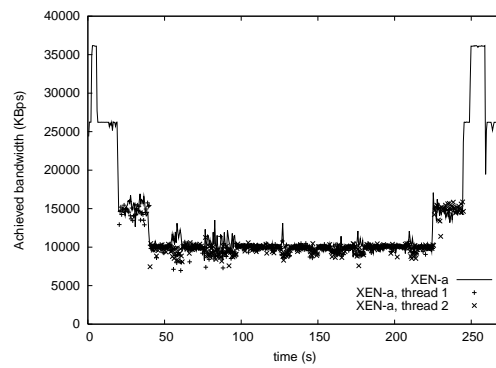


Figure 4. I/O performance of a single XEN guest running fio.



(a) Competing guests using NOOP.



(b) Competing guests using CFQ.

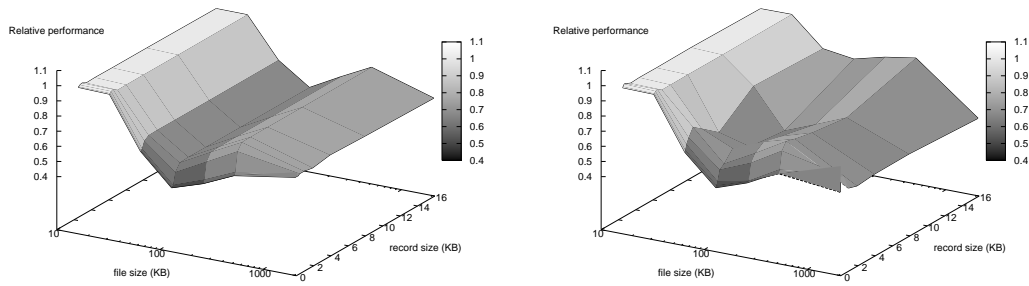
Figure 5. I/O performance of competing XEN guests running fio.

respectively. When the XEN-b processes finish, the I/O rates of XEN-a increases. The results for the CFQ experiments show the same trends. Again, the performance of XEN-a is strongly influenced by the competing traffic in XEN-b, i.e., the XEN-a rate decreases and increases when the XEN-b processes start and finish, respectively. We also observe that CFQ provides, as expected, more fair distribution of the available I/O resources compared to NOOP, but the results show in general that there is little or no performance isolation. These are surprising results, since VMs often are touted to provide performance isolation.

4.2. Degraded I/O performance of XEN

There are several studies reporting high overheads with virtualization. The reasons for the costs are not well documented or understood, but some early profiling indicates that the cost of cache and TLB misses contributes greatly to the total I/O performance degradation [27]. Buffering, copying, and synchronization between the host and the guest VM also add extra overhead [34]. Additionally, Menon et. al. [27] demonstrate an overhead (with respect to network throughput) using XEN-domain0 compared to a plain Linux kernel which is contradictory to what is reported by Barham et. al. [4].

To evaluate the I/O performance of XEN, we ran the Iozone (version 3.283) benchmark to simulate a video streaming scenario on a XEN enabled Linux 2.6.11.4 kernel. In this experiment, we did not need the fine-granular reporting of fio, but only wanted the total achievable I/O rates when retrieving large amounts of data. In this respect, Iozone can perform operations on both small and large files with small and large I/O operations, i.e., reflecting all the different data dissemination types used in video streaming (and other scenarios) – using



(a) Read.

(b) Random read.

Figure 6. Relative read performance of XEN versus Dom0 using Iozone.

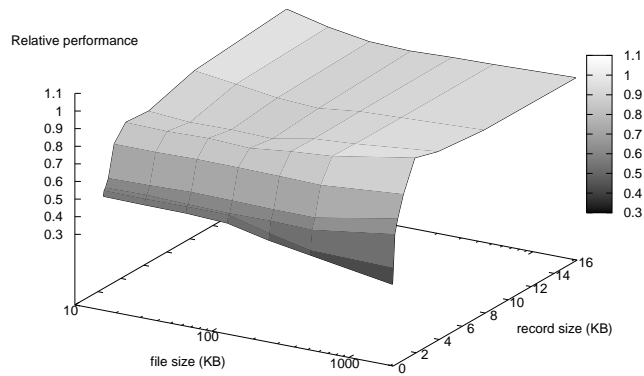


Figure 7. Relative write performance of XEN versus Dom0 using Iozone.

many small files (segments) like in the torrent-like HTTP approach used in DAVVI or using one large file sending smaller chunks in a timely manner as done using the RTSP/RTP-based Kommsys video server [41] used in our later experiments. We used Iozone in automatic mode using the *write/re-write*, *read/re-read* and *random read/write* experiments[¶]. The experiments were performed in both the host domain and in the virtual domain. Both environments had 1 GB of physical memory, and data was read from the same Seagate X15 (15K RPM) physical disk device, set up with the XFS file system. We used file sizes up to 2 GB and record sizes up to 16 KB. Each experiment was repeated 10 times, and we present the median values.

The results on the host (Dom0) and in the VM (Figure 6) show the relative performance of XEN compared to running the benchmark on the host without a virtual machine. Values below 1 indicate a VM I/O performance degradation. Figures 6(a) and 6(b) show that when reading data, there is generally a penalty using XEN. The performance is more or less the same using small files, but as the file size grows above 64 KB, we observe a large I/O performance penalty. For example, for a file size of 128 KB, if we read a very small amount of data per operation (64 B), we observe a large performance degradation of about 50%. If we read 4 KB blocks, the performance degradation is about 40%. In general, with file sizes larger than 64 KB, our measurements show at least a performance penalty of 20% for reading data (Figure 6(a)). Random read (Figure 6(b)) shows similar trends, but with a bit more variance due to the randomness.

[¶]`iozone -a [-o] -g 2G -i 0 -i 1 -i 2`

The write results (see Figure 7) show similar overhead. As Linux often performs asynchronous writes (even though the application may specify a synchronous operation), we have mounted the device with the `sync` and `dirsync` options, as well as instructing Izone to use the `O_SYNC` flag. As we can observe from the figure, XEN performance does generally degrade for write operations. Having small files and large write operations, the performance is almost equal, but small writes are heavily penalized with reductions of up to 65%.

The benchmark results show several different performance aspects of our system, and we could provide a deeper analysis of the different numbers pointing at caching/buffering effects, long data paths, added instructions, etc. However, our aim is to show whether we can observe a performance penalty using VMs or not. The experiments confirmed our expectations – the I/O performance is decreased in the XEN guest virtual machines. In our media house scenario, I/O calls will dominate performing operations on large video files, and the performance degradation may thus give severe penalties in service quality. Other VMs might be slightly better and have a smaller overhead [34], but the general observation is that virtualization is expensive.

4.3. Is full virtualization always necessary?

The advantages of running VMs to separate applications and enable multiple platforms are many, e.g., to dynamically assign resources, migrate tasks, and run different operating systems. The list is long, and the current trend seems to be to support a growing and wider set of services. However, there are several scenarios where the full isolation and protection needed in an Internet cloud are not necessary. One example is the enterprise cloud in our media house scenario where a single company may consolidate many (similar) services in an in-house cloud

running in a trusted environment. Large, full fledged VMs might be too complex and give too much overhead compared to what is necessary in this situation.

This indicates that it is important to differentiate between the Internet and enterprise consolidation scenarios, because the virtualization does not, as we demonstrated in the above sections, come for free. The requirements are different, and in media houses, many applications run on the same operating system and may be only I/O bound, e.g., running different, but functionally equal services. Thus, the bottlenecks are accesses to I/O devices and transfers of huge amounts of data over buses. Consequently, we do not need all the functionality of VMs, and considering that I/O throughput may suffer severely from virtualization, we conclude that VMs are too heavy-weight for applications only requiring I/O isolation.

Based on our experimental results, we have identified some key properties for in-house enterprise clouds: 1) *Performance isolation* must be supported, because user-perceived service quality will strongly degrade if resource availability is influenced by load peaks in other services; 2) The solution must have *low-overhead* to keep costs down and to be able to support a maximum number of users on a given set of resources; and 3) In order to support old and existing applications already installed on thousands of enterprise machines, the system should be *backward compatible* and require no changes to the applications or the operating system.

5. BONSAI

Our experience using VMs in practice indicates that there are scenarios where existing virtualization techniques are too complex and heavy weight. One example is our media house environment, where all services are controlled by a single provider (Schibsted) and where

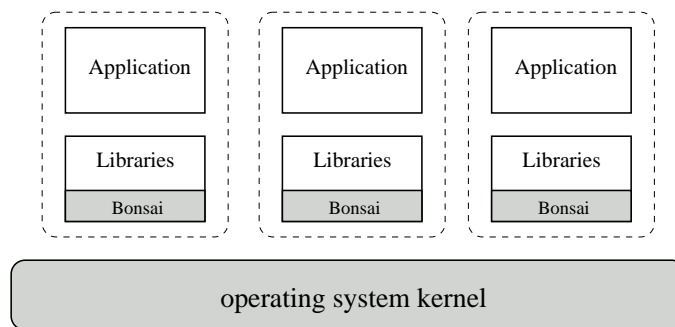


Figure 8. BONSAI architecture.

the bottleneck resource is I/O. To provide a more appropriate solution for such enterprise clouds running I/O bound applications, our BONSAI system targets the three key properties listed above in section 4.3. It is an I/O rate limitation system which is built as a light-weight alternative to large, complex, and resource consuming systems providing resource isolation. It is backward compatible and does not require modifications to applications and operating systems. BONSAI is a thin, per-application instrumentation layer, as shown by the architecture depicted in Figure 8, that transparently intercepts and subjects system calls to rate limiting. The current proof-of-concept prototype allows a system administrator to individually configure each application with a maximum rate at which the application may perform system calls. This configuration is at the granularity of individual system calls. For system calls involving I/O, such as read, write, and send, a rate is specified as the maximum number of bytes transferred per second. For all other calls, the rate is specified as the maximum number of calls per second. Thus, in a trusted environment, the system can be configured to provide the necessary isolation.

Resource assignment and limitation are enforced using a token bucket. This approach allows us to add a rate of t tokens, according to the specified limit, every period (of 1 second) and possibly support bursts of bucket size b (but for now the bucket size equals the rate). Furthermore, BONSAI is currently implemented as a modification to *glibc* version 2.4. The alternatives are adding similar functionality in the kernel or in the applications. However, both these approaches are more intrusive to the existing system. A kernel approach will require operating system updates and possibly a complete system re-installation. An application level solution forces re-implementation of the applications and allows application developers an easy way to cheat the system (by not adding the functionality). In the current solution, applications need only to dynamically link the modified *glibc* to use BONSAI. This can be done by replacing existing *glibc* libraries, modifying linking related environment variables before starting the application, or through the Linux LDPreload system. Statically linked applications must be recompiled to make use of BONSAI.

As shown in Figures 9 and 10, the modified *glibc* is set up to invoke a BONSAI handler before and after each system call. When receiving control, BONSAI inspects the call stack to discover what system call the application is attempting to perform, and in the case of I/O related calls, the amount of data involved. For each system call, BONSAI keeps track of the number of calls or amount of data transferred over time. If BONSAI detects violations of the allowed rate for a system call, the call is simply blocked until eligible. For short duration blocking, a busy wait loop is used, and for longer periods, blocking is implemented through the `select` system call. The threshold for busy wait is 1 ms in the current prototype, which after experimentation appears to be a reasonable tradeoff between the overheads of wasting CPU cycles in a busy

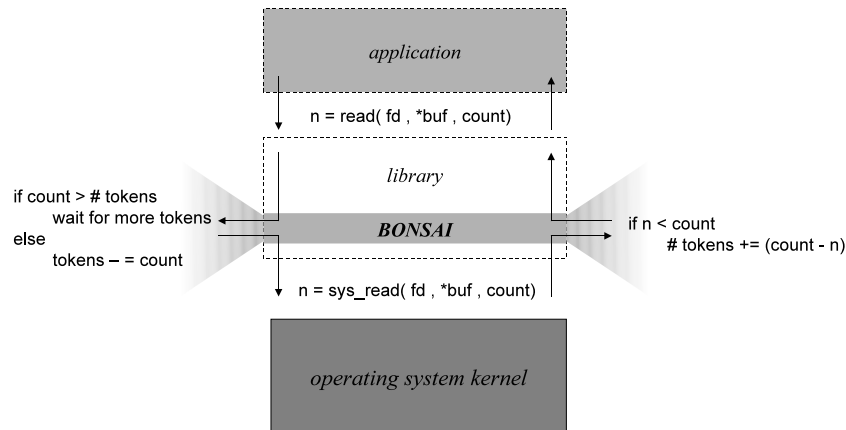


Figure 9. System call interception with `read` as an example.

wait loop and the overhead of context switching putting a process to sleep and waking it up, i.e., the results presented later in section 6.2 shows that the overhead is negligible (our main bottleneck is disk I/O, not the CPU). Furthermore, for I/O related calls, there can be a discrepancy between the requested amount of I/O and the actual amount of data processed. In particular, this is common when an application uses non-blocking I/O. To account for such discrepancies, BONSAI is invoked on the system call return path and statistics are adjusted.

6. Experiments and results

To investigate the effectiveness of BONSAI, we conducted several experiments running Linux version 2.6.15. Below, we present the results from various scenarios designed to evaluate if BONSAI might be an alternative to full virtualization in scenarios where only some light-weight I/O isolation is required.

Intercept to kernel:

```
<locate resource allotments for type of system call>
<if call cannot be completed without exceeding resource allotment>
    <calculate time till allotment has been accrued>
    <if time less than 1 milliseconds>
        <busy wait>
    <else>
        <sleep>
<continue with system call>
```

Intercept from kernel:

```
<inspect return values and account for actual resource usage>
```

Figure 10. Pseudo-code for interception of system calls.

6.1. Achieved server bandwidth

Since both Komssys and DAVVI have the same requirement for timely I/O, we ran the Komssys video server (see section 2.1) on our server machine to evaluate the potential effectiveness of BONSAI . The client processes ran on four different machines connected to the server by a Gigabit Ethernet switch. The clients requested approximately 2 Mbps constant bitrate videos, and statistics were collected on both the server and the clients.

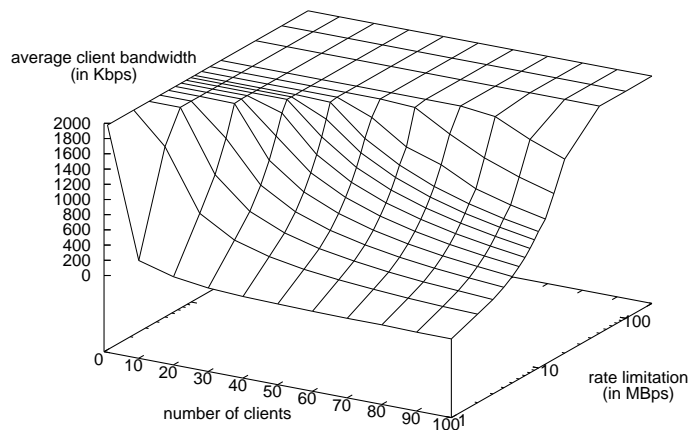


Figure 11. Average client bandwidth.

In media streaming applications, we are primarily concerned with the quality delivered to the end user. We therefore investigated the performance on the client side in a scenario where the server downgrades the service equally for all concurrent streams. In Figure 11, we show the received bandwidth at the clients in a scenario with two competing parallel server processes. The bandwidth limitations and the offered load in this scenario were varied. The maximum bandwidth per stream was 1986 Kbps, and up to 100 streams were sent to each client machine. As the Komssys video server distributes the available resources equally onto the connected clients, we observe a drop in the received bandwidth per stream when the number of streams increases, i.e., the Komssys server will start to drop quality layers in the case of a layered video file, and in the case of using DAVVI, the client system will start downloading lower quality

segments. For instance, with a rate limitation of 10 MBps (80 Mbps), a sustainable full rate is achieved for up to 40 concurrent clients, but when the number of clients increases further, the average rate drops to about 1600, 1140 and 800 Kbps for 50, 70 and 100 users, respectively. Similarly, when the amount of available resources decreases (a lower rate limitation), the received bandwidth per client drops as well. Thus, in such a scenario, these experiments show that I/O performance isolation (or only rate limitation) will be valuable to make sure that each server (and their clients) will receive the required and allocated share of the I/O resources.

As a proof of concept to gain practical experience of the BONSAI rate limitation, we have evaluated different scenarios. Figure 12 presents a representative example where the servers receive a total of 50 concurrent requests with a total read and send requirement of approximately 12.5 MBps. All the experiments show that BONSAI is able to limit the resource consumption of an application. In Figure 12(a), we have one server running without limitations, possibly stealing resources from other applications, and in Figure 12(b), the server's I/O rate in terms of number of bytes read and sent is limited to 10 MBps. We observe that the server without means for rate limiting is able to read and send all the requested data at a data rate of 12.5 MBps (Figure 12(a)). The server to which we have applied the BONSAI rate limitations is not able to read more than the upper threshold at 10 MBps (Figure 12(b), the requested rate is shown by the dotted line), which means that either the bandwidth to each client must be lowered or the server must deny some of the requests.

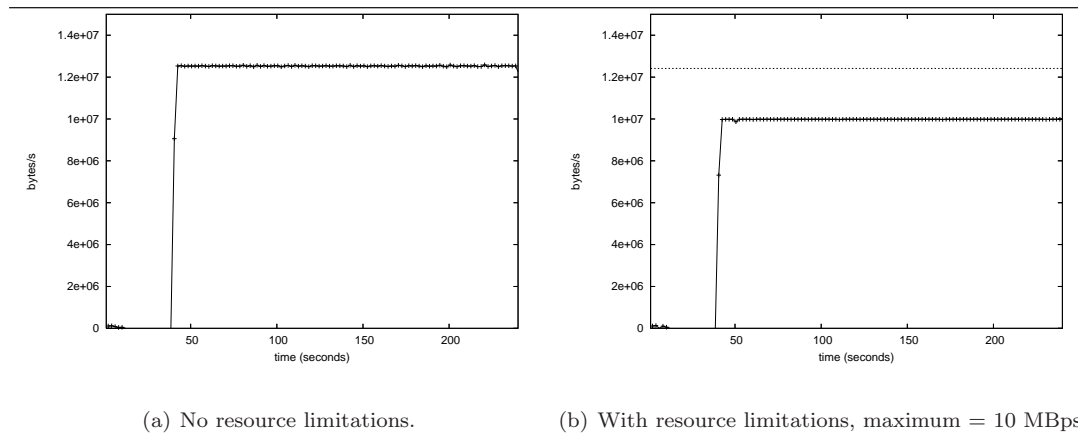


Figure 12. Aggregated I/O bandwidth on server 1 with and without limitations.

6.2. System call interception overhead

Our experiments confirm that BONSAI is able to perform rate limiting, and thus the required level of I/O isolation for enterprise consolidation. However, the trade-off is that system calls are *slightly* more expensive as additional work must be performed. For an empty system call, which is the worst case comparison, the measured overhead is ~ 550 cycles per call, which is a 48% increase over the empty call (on our Xeon machine). However, the relative overhead decreases for all other calls. In our I/O bound scenario, we will mostly perform calls like `read` and `write` where the cost also depends on the amount of data to process, i.e., in our media house scenario, the data rates are high. For example, a `write` call for a 4 KB data block might cost up to 40.000 cycles [31] (which again will be larger when the amount of data spans several disk blocks), i.e., giving a relative overhead of about 1%. So, in practice, the overhead of the important system calls are very small, and in a media-house scenario, negligible.

To confirm whether the BONSAI overhead is negligible or not, and to observe the end-to-end performance, we performed macro benchmarks using many concurrent users streaming with and without BONSAI. In our video server experiment (section 6.1), running 10 concurrent servers and varying the number of client streams from 80 to 800, we did not get any observable reduction in the I/O rate between servers running with and without BONSAI. The overhead in terms of a slightly higher CPU cycle consumption is therefore negligible in I/O-bound services. This shows the real applicability of our system and implies that the same end-to-end performance is sustained even after adding the small overhead of BONSAI per system calls.

7. Discussion

VMs enable consolidation of multiple services that possibly run their own operating system and can potentially reduce cost and management demands. Internet companies can thus provide a blend of on-demand services with a single point of access with predictable service level agreements.

In such cloud consolidation scenarios, services are often executed in different VMs that run entire guest operating systems for each service or application. In many situations, this is a convenient solution providing protection in a sandbox-like environment, migration of jobs, dynamic resource allocation, etc. For instance, for both 1) trusted private clouds where flexibility and manageability are more important than tuning the performance to a maximum, and 2) for untrusted environments like Internet clouds or external third-party clouds where the full isolation is needed to avoid interference from other, possibly malicious actors and where a VM migration makes adaption to oscillating resource requirements easier. However,

our experience is that, if high performance is a key factor, it might be beneficial to differentiate between these general Internet clouds and emerging in-house *enterprise clouds* running in a closed, trusted environment, possibly running on the same operating system and sharing data between applications. One such scenario is large media houses like our project partner Schibsted providing numerous online media services all over Europe, i.e., where the number of served clients are directly influenced by the performance of the system.

The main reason for making this distinction is that these complete VMs provide a complex set of services and add significant costs, but may not always provide the required functionality. We have shown that hypervisors like XEN may lack I/O performance isolation (see also [16, 33, 26]) and degrade I/O performance of the system (see also [27, 34]). Furthermore, a challenge is that the current trend is to add more and more functionality, and thus probably overhead, whereas the workloads in several scenarios, like the in-house media enterprise, require only a small set of the available functions that large VMs provide. One example is our media streaming consolidation scenario where most services will be I/O bound only. On the other hand, we also see more light-weight configurable VMs, which is a more interesting approach. Nevertheless, running a complete VM with its own instance of the operating system and application may be excessive and too heavy-weight. With respect to performance and resource utilization, the introduced overhead is a step in the opposite direction of the vast amount of research over the last 15-20 years in the area of optimizing systems for I/O bound workloads.

In this respect, BONSAI provides the required isolation functionality in a light-weight, backward-compatible way. Through a small library addition, commodity systems have a transparent mechanism to support rate limitations. In the enterprise setting, there is also a

question of trust and security among the providers and the customers. In our initial prototype, we have assumed a trusted environment where the provider has control over the data and applications. However, a broader scenario opens for customers to run their applications in a hosting facility where the resource provider has limited control. In such a scenario, the BONSAI prototype may therefore be too light-weight. Thus, there exists different scenarios where traditional VMs provide the best support, and others where BONSAI is best suited. The two approaches should co-exist, because they are orthogonal and may be complementary. They can therefore be used separately or combined depending on the service requirements.

One of the goals of BONSAI is to be transparent and backward-compatible, which lead to a library level implementation. However, such mechanisms can be implemented at different levels in the software stack, e.g., at the application layer, at the virtual machine layer, or at the operating system layer. We have earlier implemented a complete performance isolation kernel^{||}, and we are looking at other alternatives to the library solution. A challenge in this respect is backward compatibility. We are particularly concerned with this since, in the larger picture, we are investigating system problems for next generation distributed run-time environments, and large enterprises will probably need to be able to run some of their existing software components.

^{||}See Vortex - <http://www.iad.cs.wit.no/>

8. Related work

Much research has addressed guaranteed access to resources and meeting QoS goals by admission control. At the operating system level, all services may be forced through the same admission mechanisms to provide system wide control. In our streaming server example, this may include limits on the total rate or number of streams using admission control and service differentiation based on connection and application level information [38]. Similarly, admission to network resources may be granted based on available CPU cycles [25], bandwidth [5], buffers [36], and measured latencies [15]. Admission control can alternatively be placed in a middleware [6, 20] or in the application [3, 7].

More advanced systems include dynamic schedulers with a notion of admission control, e.g., the CPU schedulers in RT-MACH [28], Resource Kernels [32], and Rialto [23]. Here, resource accounting for a process is a challenge, but systems like Eclipse [9] have support for provisioning and reservation of resources. However, provisioning and reservation tasks are complex, and to simplify them, operating system code may be moved to user-level libraries like in Nemesis [18] (and similarly in Cache Kernel [10] and Exokernel [14]). Furthermore, many mechanisms have been proposed to better utilize the resources and give guarantees to the users by looking at disk bandwidth, I/O requests, buffers, or response times [37, 40]. A similar approach as ours, and concurrently developed with our system, is the I/O scheduler presented in [12], using similar ideas with resource accounting.

Moreover, several commercial operating systems like HP-UX [17], z/OS [19] and Solaris [35] include frameworks for management of resources, but these systems focus on long-term goals

and rely on fair-share scheduling approaches for enforcement of resource shares. Resources that cannot be replenished (such as disk space) are typically controlled by hard limits.

However, all the mechanisms listed above require (possibly large) changes to the operating system with necessary schedulers and instrumentation to attribute resource usage to the various services. Thus, many of the proposed components are not even implemented but only analytically evaluated or simulated.

Virtualization systems like IBM VM/370 [11] and XEN [4] partition the machine resources in an isolated way. However, a complete virtualization system may be too complex and introduce overhead that reduces the overall performance. For example, the results presented for XEN show some small performance penalties over 'bare metal' Linux [4], but for server-type workloads, "XEN lags an unvirtualized system by up to 38% for network throughput while demanding a comparable CPU load, and 25% for disk intensive workloads" [34]. In many consolidation scenarios, I/O resources are the main bottlenecks, and isolation by virtualization may incur noticeable performance reductions. Furthermore, there are also work addressing the lack of resource isolation. For example, in XEN, the Self Earliest Deadline First - Debt Collector (SEDF-DC) scheduler improves the CPU resource isolation whereas ShareGuard enhances the isolation of the network resources [16]. However, it is unclear whether disk resources now are supported (as stated in future work), and whether they still will experience the overheads of a complete VM.

Our approach, on the other hand, does not rely on changes to the applications or the operating system, and introduces a minimum of overhead that does not noticeably reduce I/O

performance. Performance isolation is transparently achieved by intercepting and applying traffic shaping to system calls.

9. Conclusion

Virtualization has become immensely popular, and in a service consolidation scenario, like the emerging cloud infrastructures, VMs are the de facto solution. Although this technology is very useful in many scenarios, the touting of VMs as an universal solution makes people jump too easily for such a solution without considering the drawbacks and possible alternatives. For example, we have demonstrated that hypervisors lack I/O performance isolation and degrade I/O performance of the system. Thus, for consolidated I/O-bound services, a complete VM with an own instance of the operating system for each application may be too excessive and heavy weight.

Furthermore, we have argued that the necessary functionality for consolidating this type of service can be provided in a more cost-efficient and backward compatible way. Empirical data so far indicates that the required level of performance isolation can be achieved by modifying system calls and intercepting the calls in user space libraries, i.e., providing the I/O isolation for the application-level services in a transparent way, just as existing VMs. This thin instrumentation layer monitors the rate of every data stream using a per-stream token bucket, and if the requested rate exceeds the specified limit, the call is blocked until there is available tokens. For the presented experiments, resource limits have been hand-tuned for the proof-of-concept, and research into automatic management is currently ongoing. We are also currently working on schemes where soft guarantees are given to resource consumers as a rule,

but where the upside is potential and free support for more exceptional resource demands. In this way, resource consumers can plan for the common case and have some assurance that the more exceptional case still can be supported.

ACKNOWLEDGEMENTS

This work has been performed in the context of the *iAD* (Information Access Disruptions) centre for Research-based Innovation, project number 174867. *iAD* is directed by Fast Search & Transfer in collaboration with Schibsted, one of Europe's largest media companies, Accenture, Cornell University, University College Dublin, Dublin City University, BI Norwegian School of Management and the Norwegian universities in Tromsø (UiT), Trondheim (NTNU) and Oslo (UiO). In particular, we are grateful for several in-dept discussions with principal investigator Erik Aaberg in Schibsted.

REFERENCES

1. Move Networks. <http://www.movenetworks.com>, 2009.
2. SmoothHD. <http://www.smoothhd.com>, 2009.
3. ALMEIDA, J., DABU, M., MANIKUTTY, A., AND CAO, P. Providing differentiated levels of service in web content hosting. In *Proceedings of the ACM SIGMETRICS workshop on Internet server performance (WISP)* (Madison, Wisconsin, June 1998), pp. 91–102.
4. BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. In *Proceedings of the symposium on operating system principles (SOSP)* (Bolton Landing, NY, October 2003), pp. 164–177.
5. BENAMEUR, N., FREDJ, S. B., DELCOIGNE, F., OUESLATI-BOULAHIA, S., AND ROBERTS, J. W. Integrated admission control for streaming and elastic traffic. In *Proceedings of the international workshop on quality of future Internet services (QofIS)* (London, UK, 2001), pp. 69–81.

-
6. BHATTI, N., AND FRIEDRICH, R. Web server support for tiered services. *IEEE network* 13, 5 (September 1999), 64–71.
 7. BHOJ, P., RAMANATHAN, S., AND SINGHAL, S. Web2k: Bringing QoS to web servers. Tech. Rep. HPL-2000-61, Hewlet Packard, May 2000.
 8. BOEHM, H.-J., ATKINSON, R., AND PLASS, M. Ropes: An alternative to strings. *Software: Practice and Experience* 25, 12 (1995), 1315–1330.
 9. BRUNO, J., GABBER, E., OZDEN, B., AND SILBERSCHATZ, A. The Eclipse operating system: Providing quality of service via reservation domains. In *Proceedings of USENIX annual technical conference* (New Orleans, Louisiana, USA, June 1998), pp. 235–246.
 10. CHERITON, D. R., AND DUDA, K. J. A caching model of operating system functionality. In *Proceedings of the symposium on Operating Systems Design and Implementation (OSDI)* (Monterey, CA, November 1994), pp. 179–193.
 11. CHESS, D. M., AND WALDBAUM, G. The VM/370 resource limiter. *IBM Systems Journal* 20, 4 (1981), 424–437.
 12. CRACIUNAS, S. S., KIRSCH, C. M., AND RÖCK, H. I/o resource management through system call scheduling. *SIGOPS Operating System Review* 42, 5 (2008), 44–54.
 13. DEVINE, S. W., BUGNION, E., AND ROSENBLUM, M. Virtualization system including a virtual machine monitor for a computer with a segmented architecture. United States Patent 6397242, Oct. 1998.
 14. ENGLER, D., KAASHOEK, M., AND O'TOOLE JR., J. Exokernel: An operating system architecture for application-level resource management. In *Proceedings of the ACM symposium on operating systems principles (SOSP)* (Copper Mountain Resort, Colorado, USA, December 1995), pp. 251–266.
 15. GOPALAN, K., HUANG, L., PENG, G., CHIUH, T.-C., AND LIN, Y.-J. Statistical admission control using delay distribution measurements. *ACM Transactions on Multimedia Computing, Communications and Applications* 2, 4 (2006), 258–281.
 16. GUPTA, D., CHERKASOVA, L., GARDNER, R., AND VAHDAT, A. Enforcing performance isolation across virtual machines in xen. In *Proceedings of the international middleware conference (Middleware)* (Melbourne, Australia, 2006), pp. 342–362.
 17. HP-UX WORLOAD MANAGER. <http://h30081.www3.hp.com/products/wlm/index.html>.
 18. HYDEN, E. *Operating system support for quality of service*. PhD thesis, Computer Laboratory, University
-

-
- of Cambridge, February 1994.
19. IBM z/OS WORKLOAD MANAGER. <http://www-1.ibm.com/servers/eserver/zseries/zos/wlm/>, 2007.
 20. JAMJOOM, H., AND REUMANN, J. Qguard: Protecting internet servers from overload. Tech. Rep. CSE-TR-427-00, University of Michigan, 2000.
 21. JOHANSEN, D., JOHANSEN, H., AARFLOT, T., HURLEY, J., KVALNES, Å., GURRIN, C., SAV, S., OLSTAD, B., AABERG, E., ENDESTAD, T., RIISER, H., GRIWODZ, C., AND HALVORSEN, P. DAVVI: A prototype for the next generation multimedia entertainment platform. In *Proceedings of ACM International Multimedia Conference (ACM MM)* (Oct. 2009), pp. 989–990.
 22. JOHNSEN, F. T., HAFSØE, T., GRIWODZ, C., AND HALVORSEN, P. Workload characterization for news-on-demand streaming services. In *Proceedings of the IEEE international performance computing and communications conference (IPCCC)* (New Orleans, LA, USA, Apr. 2007).
 23. JONES, M. B., ROSU, D., AND ROSU, M.-C. CPU reservations and time constraints: efficient, predictable scheduling of independent activities. In *Proceedings of the ACM symposium on operating systems principles (SOSP)* (Saint Malo, France, October 1997), pp. 198–211.
 24. KÄPPNER, T., AND WOLF, L. C. Media scaling in distributed multimedia object services. In *IWACA '94: Proceedings of the Second International Workshop on Multimedia* (London, UK, 1994), Springer-Verlag, pp. 34–43.
 25. KIM, K., AND NAHRSTEDT, K. Qos translation and admission control for mpeg video. In *Proceedings of the international workshop on quality of service (IWQoS)* (San Francisco, CA, USA, 1997), pp. 359–362.
 26. MATTHEWS, J. N., HU, W., HAPUARACHCHI, M., DESHANE, T., DIMATOS, D., HAMILTON, G., MCCABE, M., AND OWENS, J. Quantifying the performance isolation properties of virtualization systems. In *Proceedings of the workshop on experimental computer science (ExpCS)* (San Diego, CA, USA, 2007), p. 6.
 27. MENON, A., SANTOS, J. R., TURNER, Y., JANAKIRAMAN, G. J., AND ZWAENEPOEL, W. Diagnosing performance overheads in the xen virtual machine environment. In *Proceedings of the ACM/USENIX international conference on virtual execution environments (VEE)* (Chicago, IL, USA, 2005), pp. 13–23.
 28. MERCER, C. W., SAVAGE, S., AND TOKUDA, H. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of the IEEE international conference on multimedia computing and systems (ICMCS)* (Boston, MA, USA, May 1994), pp. 90–99.
 29. NURMI, D., WOLSKI, R., GRZEGORCZYK, C., OBERTELLI, G., SOMAN, S., YOUSEFF, L., AND ZAGORODNOV,
-

-
- D. Eucalyptus: A technical report on an elastic utility computing architecture linking your programs to useful systems. Tech. Rep. 2008-10, University of California, Santa Barbara, Department of Computer Science, aug 2008.
30. PERILLI, A. Amazon launches xen-powered virtual datacenter on demand, <http://www.virtualization.info/2006/08/amazon-launches-xen-powered-virtual.html>.
31. RAJAGOPALAN, M., HILTUNEN, M., JIM, T., AND SCHLICHTING, R. Authenticated system calls. In *Proceedings of the international conference on dependable systems and networks (DSN)* (Yokohama, Japan, June 2005), pp. 358–367.
32. RAJKUMAR, R., JUVVA, K., MOLANO, A., AND OIKAWA, S. Resource kernels: A resource-centric approach to real-time systems. In *Proceedings of the SPIE/ACM conference on multimedia computing and networking (MMCN)* (San Jose, CA, January 1998), pp. 150–164.
33. SEELAM, S. R., AND TELLER, P. J. Virtual i/o scheduler: a scheduler of schedulers for performance virtualization. In *Proceedings of the international conference on virtual execution environments (VEE)* (San Diego, CA, USA, 2007), pp. 105–115.
34. SOLTESZ, S., PÖTZL, H., FIUCZYNSKI, M. E., BAVIER, A., AND PETERSON, L. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *Proceedings of the ACM SIGOPS/EuroSys European conference on computer systems (EuroSys)* (Lisbon, Portugal, 2007), pp. 275–287.
35. SUN MICROSYSTEMS INC. Solaris Resource Manager 1.0 (white paper).
36. VENKATRAMANT, M., AND NASRABADIS, N. M. An admission control framework to support media-streaming overpacket-switched networks. In *Proceedings of the IEEE international conference on communications (ICC)* (Vancouver, BC, Canada, 1999), pp. 1357–1361.
37. VIN, H., GOYAL, P., GOYAL, A., AND GOYAL, A. A statistical admission control algorithm for multimedia servers. In *Proceedings of the ACM international conference on multimedia (MM)* (San Francisco, CA, USA, 1994), pp. 33–40.
38. VOIGT, T., TEWARI, R., FREIMUTH, D., AND MEHRA, A. Kernel mechanisms for service differentiation in overloaded web servers. In *Proceedings of the USENIX annual technical conference* (Boston, MA, USA, June 2001), pp. 189–202.
39. YU, H., ZHENG, D., ZHAO, B. Y., AND ZHENG, W. Understanding user behavior in large-scale video-
-

-
- on-demand systems. In *Proceedings of the ACM SIGOPS/EuroSys European conference on computer systems (EuroSys)* (Leuven, Belgium, 2006), pp. 333–344.
40. ZIMMERMANN, R., AND FU, K. Comprehensive statistical admission control for streaming media servers. In *Proceedings of the ACM international conference on multimedia (MM)* (New York, NY, USA, 2003), ACM Press, pp. 75–85.
41. ZINK, M., GRIWODZ, C., AND STEINMETZ, R. KOM player - a platform for experimental VoD research. In *Proceedings of the IEEE symposium on computers and communications (ISCC)* (Hammamet, Tunisia, July 2001), pp. 370–375.