


Evaluating storage APIs for QEMU

Anthony Liguori – aliguori@us.ibm.com
Open Virtualization
IBM Linux Technology Center

Linux Plumbers Conference 2009

The V-Word

- QEMU is used by Xen and KVM for I/O but...
  - this is not a virtualization talk
- Let's just think of QEMU as a userspace process that can run a variety of “workloads”
- Think of it like dbench
- These workloads tend to be very intelligent about how they access storage
- Workloads have incredible performance demands
- Our goal is to give our users the best possible performance by default
  - Should Just Work

We want

- Asynchronous completion
- Scatter/gather lists
- Batch submission
- Ability to tell kernel about request ordering requirements
- Ability to maintain CPU affinity for request processing

Hello World

Posix read()/write()

- Our very first implementation
- We handled requests synchronously, using read()/write()
- Scatter/gather lists were bounced
- Main problem with this approach:
  - Workload cannot run while processing I/O request
  - I/O performance is terrible
  - Because workload doesn't run while waiting for I/O, CPU performance is terrible too

Worker thread

First improvement

- Have a single worker thread
- I/O requests are now asynchronous
  - No more horrendous CPU overhead
- We still bounce
- We can only handle one request at a time
- Never merged upstream (Xen only)

posix-aio

Upstream solution

- Use posix-aio to support portable AIO
- Yay!
- Reasonable API
  - Can batch requests
  - Supports async notification via signals
- Except it's terrible

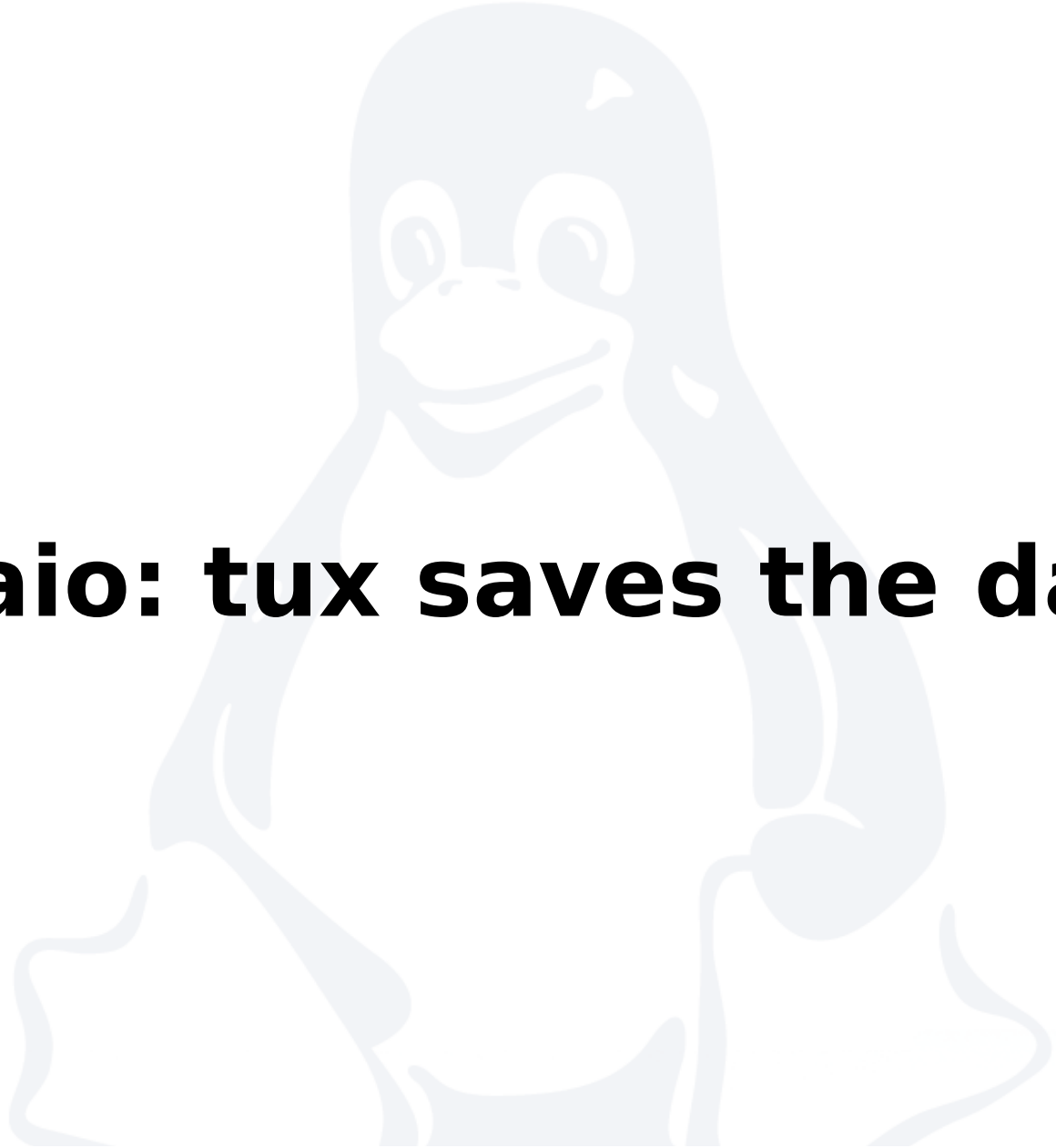
Posix-aio shortcomings

- Under the covers, it uses a thread pool
- Requires bouncing
- API is not extendable by mere mortals
  - New APIs must be accepted by POSIX before implementing in glibc (or so I was told)
- Biggest problem was this comment in glibc:
- “ The current file descriptor is worked on. It makes no sense to start another thread since this new thread would fight with the running thread for the resources.”
- Cannot support multiple AIO requests in flight on a single file descriptor; no response from Ulrich about removing this restriction
- Signal based completion is painful to use

Other posix-aio's

- It's not just glibc that screws it up
- FreeBSD has a nice posix-aio implementation that's supported by a kernel module
- If you use posix-aio without this module loaded, you get a SEGV
- You need non-portable code to detect if this kernel module is not loaded, and then a fallback mechanism that isn't posix-aio since a non-privileged user cannot load kernel modules
- Posix-aio always requires a fallback

linux-aio: tux saves the day!



linux-aio

- Forget portability, let's use a native linux interface
- Fall back to something lame for everything else
- Very nice interface
  - Supports scatter/gather requests
  - Can submit multiple requests at once
- Except it's terrible

linux-aio shortcomings

- Originally, no async notification
  - Must use special blocking function
  - Signal support added
  - Eventfd support added
  - Neither mechanism is probe-able in software so you have to guess at compile time
  - Libaio spent a good period of time in an unmaintained state making eventfd support unavailable in even modern distros (SLES11)
- Only works on some types of file descriptors
  - Usually, O_DIRECT
- If used on an unsupported file descriptor, you get no error, io_submit() just blocks

!@#!@@$#!!#@#!#@

linux-maybe-sometimes-aio

- There is no right way to use this API if you actually care about asynchronous IO requests
- You either have to
  - Require a user to enable linux-aio
  - Be extremely conversation and limit yourselves to things you know work today like O_DIRECT on a physical device
- No guarantee these cases will keep working
- No way of detecting when new cases are added
- The API desperately needs feature detection
- It's only useful for databases and benchmarking tools

Let's fix posix-aio

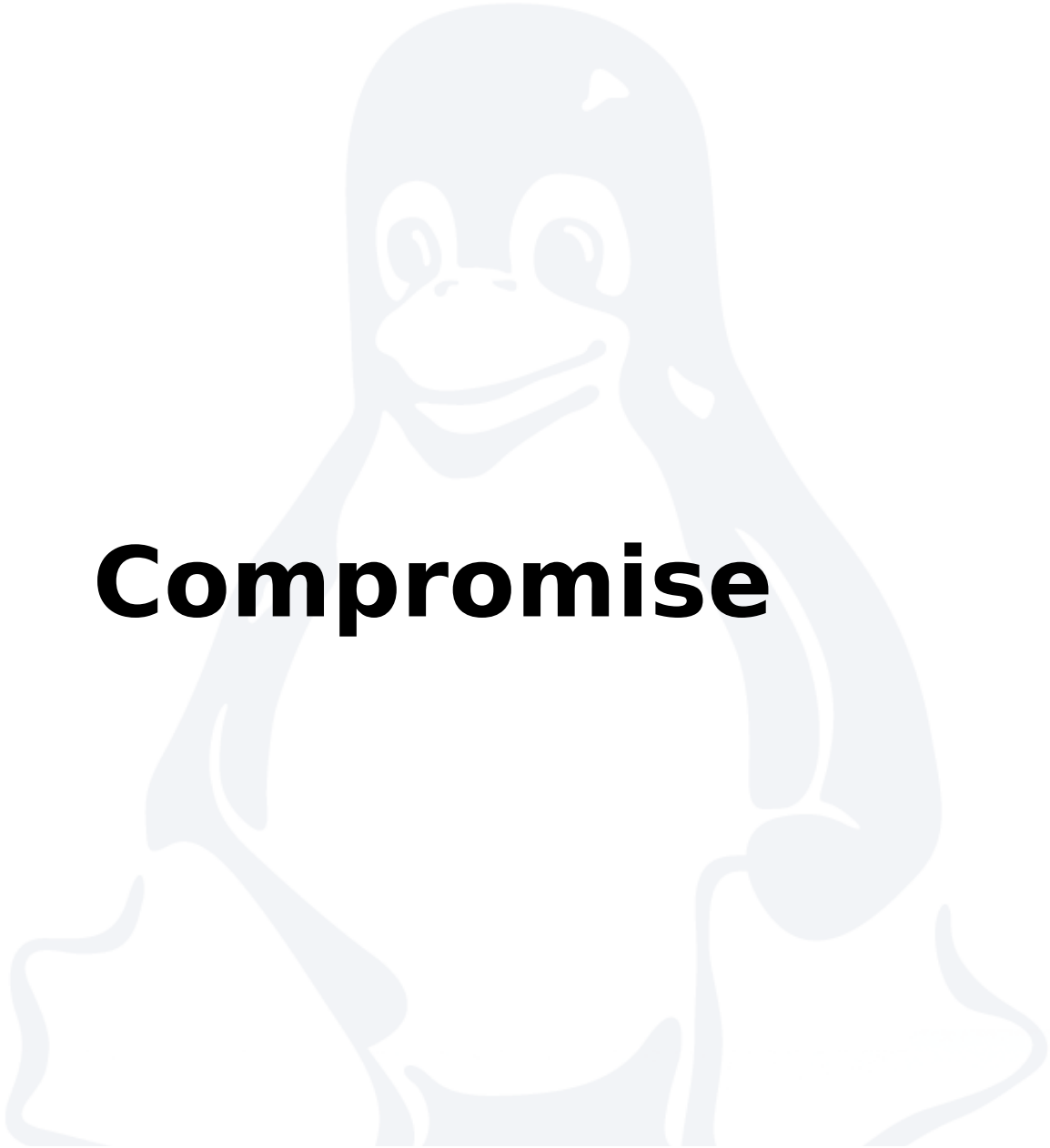
Our own thread pool

- Implement our own posix-aio but don't enforce arbitrary limits
- Still cannot submit multiple requests on a file descriptor because of seek/read race
  - Thread1: lseek -> readv
  - Thread2: lseek -> (race) -> writev
- Tried various work-arounds with dup() (FAIL)
- Bounce buffers and use pread/pwrite
- Introduce preadv/pwritev
  - We now have zero copy and simultaneous request processing

Shortcomings

- Thread switch cost is non-negligible
- We don't have a true batch submission API to the kernel
  - Tagging semantics don't map very well
- Not very CFQ friendly
  - Each thread is considered a different IO context, CFQ waits for each thread to submit more requests resulting in long delays
  - Fixable with CLONE_IO – not exposed through pthreads
  - Some attempts at improving upstream

Compromise



What we do today

- We use linux-aio when we think it's safe
  - Gives us better performance
  - Only use with block devices
  - Lose features such as host page cache sharing
  - For certain configurations, like c _ _ _ d, making use of the host page cache is absolutely critical
  - Most users use file backed images
- We fall back to our thread pool otherwise
  - Good compromise of performance and features
  - But we know we can do better

What's coming

acall/syslets

- Both are kernel thread pool
  - Avoid thread creation when request can complete immediately (nice)
  - Lighter weight threads
  - Potentially better thread pool management
- acall has a narrower scope
  - No clear benefit *today* over userspace thread pool other than introducing interfaces
  - Seems easier to merge upstream
- syslets have a broader scope
  - Complex ability to chain system calls without returning to userspace
  - Seems to have lost merge momentum

acall/syslet shortcomings

- Still does not solve some of the fundamental semantic mapping issues
  - Neither are very useful for our workloads without preadv/pwritev
  - Neither help request tagging as request ordering is fundamentally lost in a thread pool
  - Still not obvious how to extend preadv/pwritev paradigm to support tagging
  - Both have clear benefits though

Overall uncertainty

- We're willing to fix linux-aio
- We're willing to help solve the problems around acall/syslets
- The lack of clarity around the future makes it difficult though to begin
- Other v-word solutions use custom userspace block IO interfaces to avoid these problems
  - Using confusing terms like “in-kernel paravirtual block device backend” to avoid real review
  - It would be much better to fix the generic interfaces so everyone benefits
  - It's a battle we're losing so far

Questions

- Questions?

Evaluating storage APIs for QEMU

Anthony Liguori – aliguori@us.ibm.com
Open Virtualization
IBM Linux Technology Center

Linux Plumbers Conference 2009

The V-Word

- QEMU is used by Xen and KVM for I/O but...
  - this is not a virtualization talk
- Let's just think of QEMU as a userspace process that can run a variety of “workloads”
- Think of it like dbench
- These workloads tend to be very intelligent about how they access storage
- Workloads have incredible performance demands
- Our goal is to give our users the best possible performance by default
  - Should Just Work

We want

- Asynchronous completion
- Scatter/gather lists
- Batch submission
- Ability to tell kernel about request ordering requirements
- Ability to maintain CPU affinity for request processing

Hello World

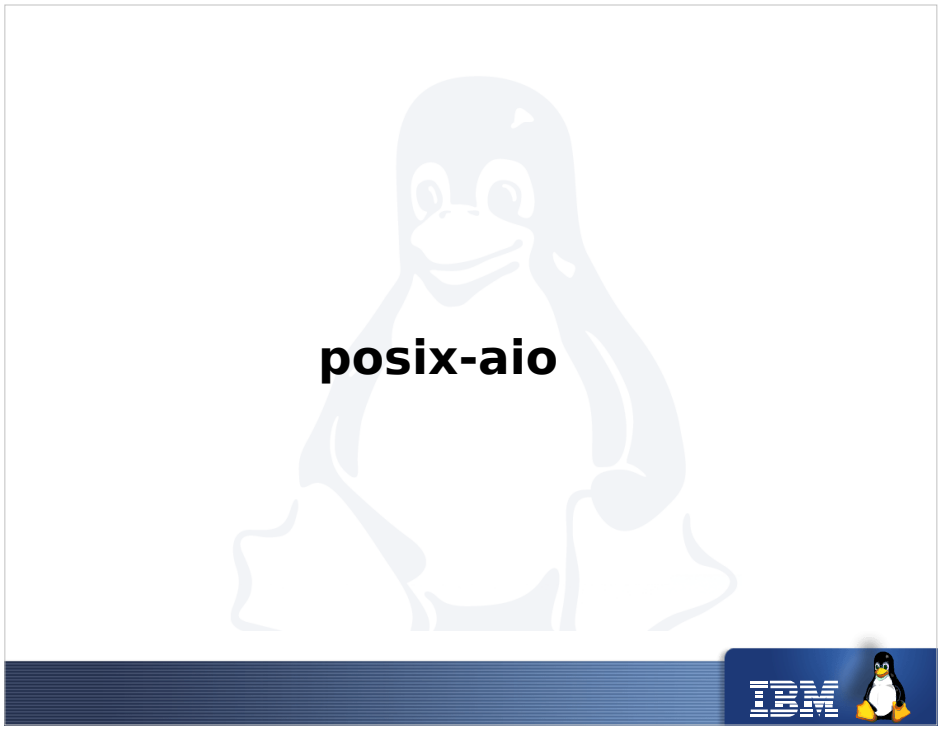
Posix read()/write()

- Our very first implementation
- We handled requests synchronously, using read()/write()
- Scatter/gather lists were bounced
- Main problem with this approach:
  - Workload cannot run while processing I/O request
  - I/O performance is terrible
  - Because workload doesn't run while waiting for I/O, CPU performance is terrible too

Worker thread

First improvement

- Have a single worker thread
- I/O requests are now asynchronous
  - No more horrendous CPU overhead
- We still bounce
- We can only handle one request at a time
- Never merged upstream (Xen only)


posix-aio
IBM

Upstream solution

- Use posix-aio to support portable AIO
- Yay!
- Reasonable API
  - Can batch requests
  - Supports async notification via signals
- Except it's terrible

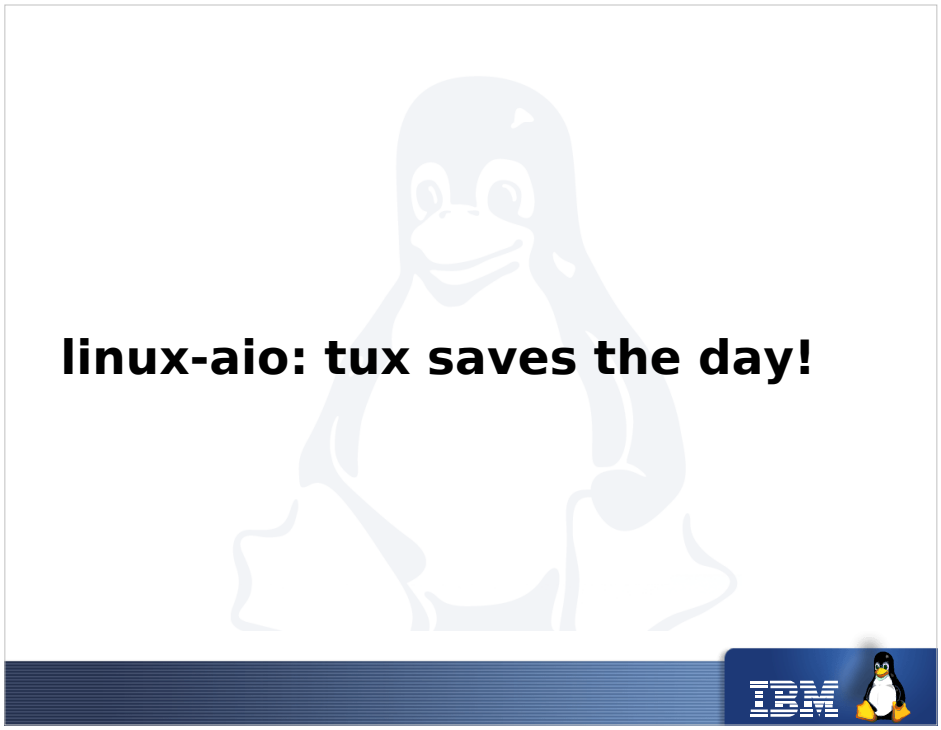
Posix-aio shortcomings

- Under the covers, it uses a thread pool
- Requires bouncing
- API is not extendable by mere mortals
  - New APIs must be accepted by POSIX before implementing in glibc (or so I was told)
- Biggest problem was this comment in glibc:
- " The current file descriptor is worked on. It makes no sense to start another thread since this new thread would fight with the running thread for the resources."
- Cannot support multiple AIO requests in flight on a single file descriptor; no response from Ulrich about removing this restriction
- Signal based completion is painful to use

Other posix-aio's

- It's not just glibc that screws it up
- FreeBSD has a nice posix-aio implementation that's supported by a kernel module
- If you use posix-aio without this module loaded, you get a SEGV
- You need non-portable code to detect if this kernel module is not loaded, and then a fallback mechanism that isn't posix-aio since a non-privileged user cannot load kernel modules
- Posix-aio always requires a fallback

linux-aio: tux saves the day!

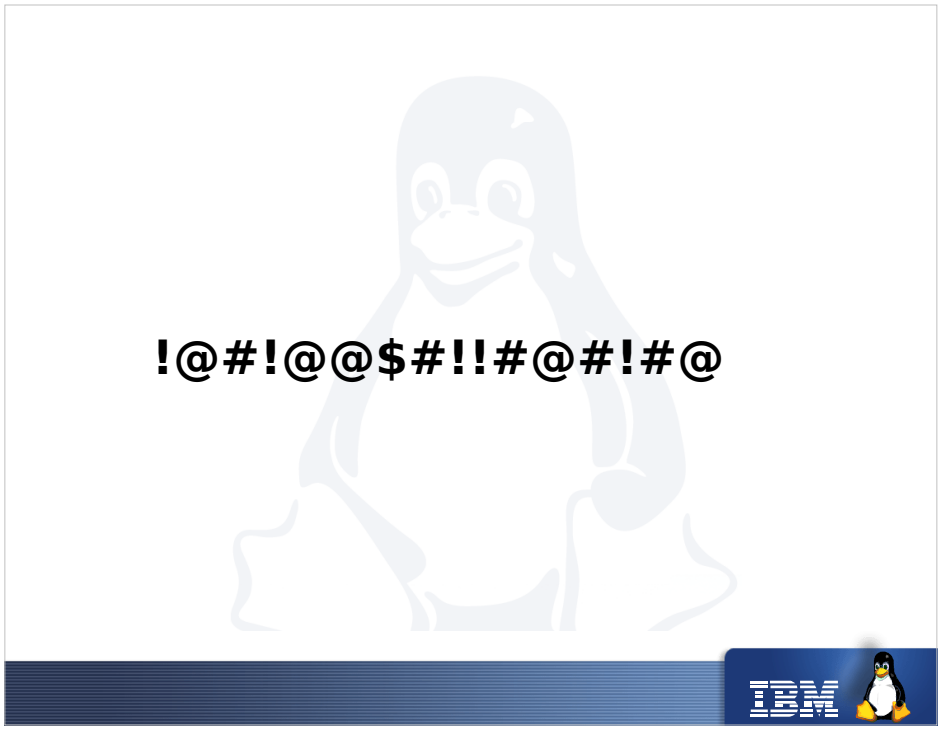


linux-aio

- Forget portability, let's use a native linux interface
- Fall back to something lame for everything else
- Very nice interface
  - Supports scatter/gather requests
  - Can submit multiple requests at once
- Except it's terrible

linux-aio shortcomings

- Originally, no async notification
  - Must use special blocking function
  - Signal support added
  - Eventfd support added
  - Neither mechanism is probe-able in software so you have to guess at compile time
  - Libaio spent a good period of time in an unmaintained state making eventfd support unavailable in even modern distros (SLES11)
- Only works on some types of file descriptors
  - Usually, O_DIRECT
- If used on an unsupported file descriptor, you get no error, io_submit() just blocks


!@#!@@$#!!#@#!#@
IBM

linux-maybe-sometimes-aio

- There is no right way to use this API if you actually care about asynchronous IO requests
- You either have to
  - Require a user to enable linux-aio
  - Be extremely conversation and limit yourselves to things you know work today like O_DIRECT on a physical device
- No guarantee these cases will keep working
- No way of detecting when new cases are added
- The API desperately needs feature detection
- It's only useful for databases and benchmarking tools

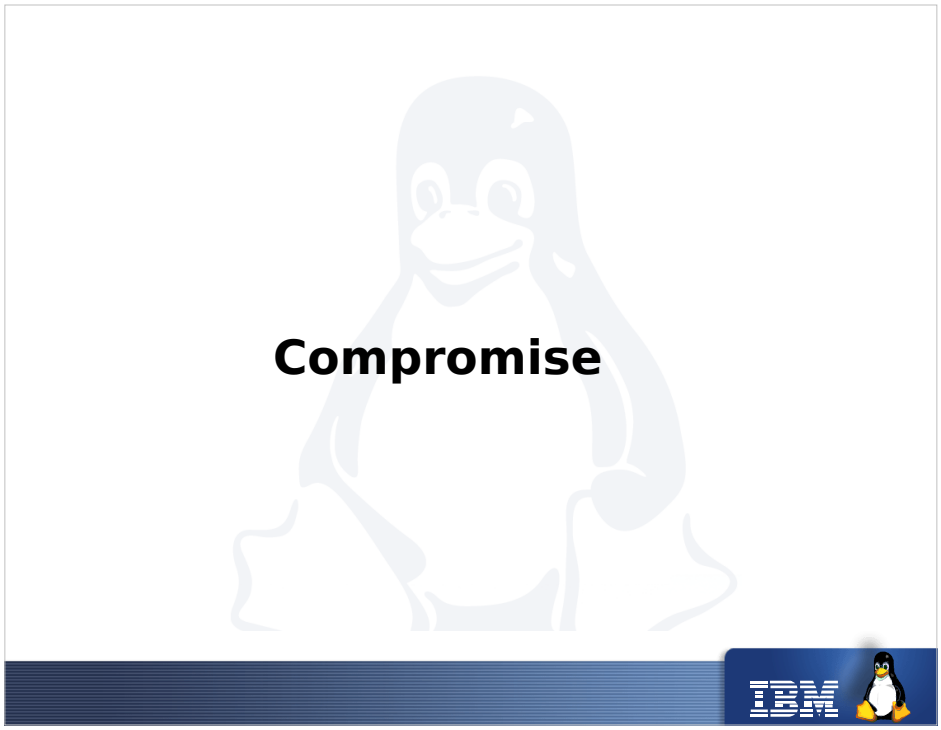
Let's fix posix-aio

Our own thread pool

- Implement our own posix-aio but don't enforce arbitrary limits
- Still cannot submit multiple requests on a file descriptor because of seek/read race
  - Thread1: lseek -> readv
  - Thread2: lseek -> (race) -> writev
- Tried various work-arounds with dup() (FAIL)
- Bounce buffers and use pread/pwrite
- Introduce preadv/pwritev
  - We now have zero copy and simultaneous request processing

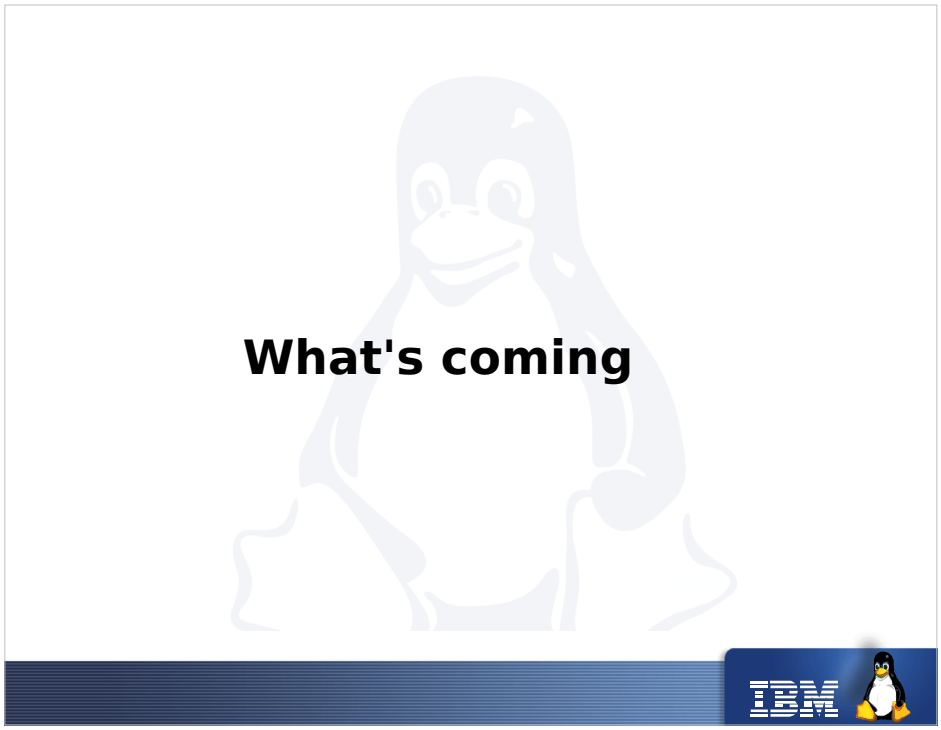
Shortcomings

- Thread switch cost is non-negligible
- We don't have a true batch submission API to the kernel
  - Tagging semantics don't map very well
- Not very CFQ friendly
  - Each thread is considered a different IO context, CFQ waits for each thread to submit more requests resulting in long delays
  - Fixable with CLONE_IO – not exposed through pthreads
  - Some attempts at improving upstream


Compromise
IBM

What we do today

- We use linux-aio when we think it's safe
  - Gives us better performance
  - Only use with block devices
  - Lose features such as host page cache sharing
  - For certain configurations, like c _ _ _ d, making use of the host page cache is absolutely critical
  - Most users use file backed images
- We fall back to our thread pool otherwise
  - Good compromise of performance and features
  - But we know we can do better


What's coming

acall/syslets

- Both are kernel thread pool
  - Avoid thread creation when request can complete immediately (nice)
  - Lighter weight threads
  - Potentially better thread pool management
- acall has a narrower scope
  - No clear benefit *today* over userspace thread pool other than introducing interfaces
  - Seems easier to merge upstream
- syslets have a broader scope
  - Complex ability to chain system calls without returning to userspace
  - Seems to have lost merge momentum

acall/syslet shortcomings

- Still does not solve some of the fundamental semantic mapping issues
  - Neither are very useful for our workloads without preadv/pwritev
  - Neither help request tagging as request ordering is fundamentally lost in a thread pool
  - Still not obvious how to extend preadv/pwritev paradigm to support tagging
  - Both have clear benefits though

Overall uncertainty

- We're willing to fix linux-aio
- We're willing to help solve the problems around acall/syslets
- The lack of clarity around the future makes it difficult though to begin
- Other v-word solutions use custom userspace block IO interfaces to avoid these problems
  - Using confusing terms like “in-kernel paravirtual block device backend” to avoid real review
  - It would be much better to fix the generic interfaces so everyone benefits
  - It's a battle we're losing so far

Questions

- Questions?