Making The Linux NFS Server Suck Faster

Greg Banks <gnb@melbourne.sgi.com>
File Serving Technologies,
Silicon Graphics, Inc

Overview

- Introduction
- Principles of Operation
- Performance Factors
- Performance Results
- Future Work
- Questions?

SGI

- SGI doesn't just make honking great compute servers
- also about storage hardware
- and storage software
- **NAS Server Software**

NAS

- **NAS** = **N**etwork **A**ttached **S**torage

NAS

- your data on a RAID array
- attached to a special-purpose machine with network interfaces

sgi INNOVATION FOR RESULTS

NAS

- Access data over the network via **file sharing** protocols
- CIFS
- iSCSI
- FTP
- **NFS**

NAS

- NFS: a common solution for compute cluster storage
- freely available
- known administration
- no inherent node limit
- simpler than cluster filesystems (CXFS, Lustre)

Anatomy of a Compute Cluster

- today's HPC architecture of choice
- hordes (2000+) of Linux 2.4 or 2.6 clients

Anatomy of a Compute Cluster

- node: low bandwidth or IOPS
- 1 Gigabit Ethernet NIC

- server: large **aggregate** bandwidth or IOPS
- multiple Gigabit Ethernet NICs...2 to 8 or more

sgi® INNOVATION FOR RESULTS

Anatomy of a Compute Cluster

- global namespace desirable
- sometimes, a single filesystem
- Ethernet bonding or RR-DNS

SGI's NAS Server

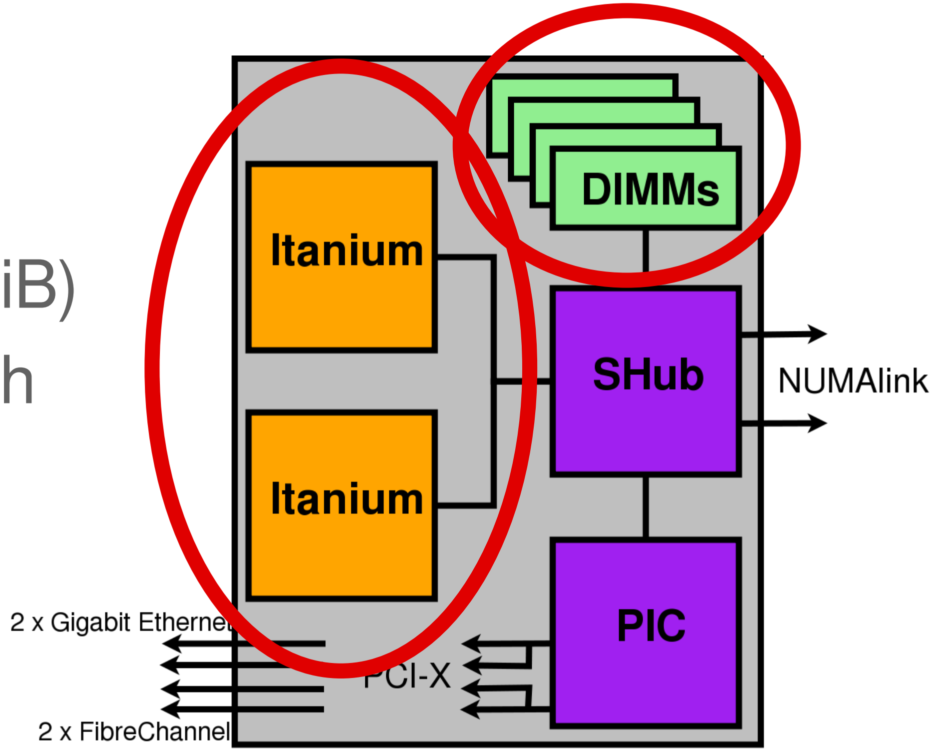
- SGI's approach: a single honking great server
- global namespace happens trivially
- large RAM fit
- shared data & metadata cache
- performance by scaling UP not OUT

SGI's NAS Server

- IA64 Linux NUMA machines (Altix)
- previous generation: MIPS Irix (Origin)
- small by SGI's standards (2 to 8 CPUs)

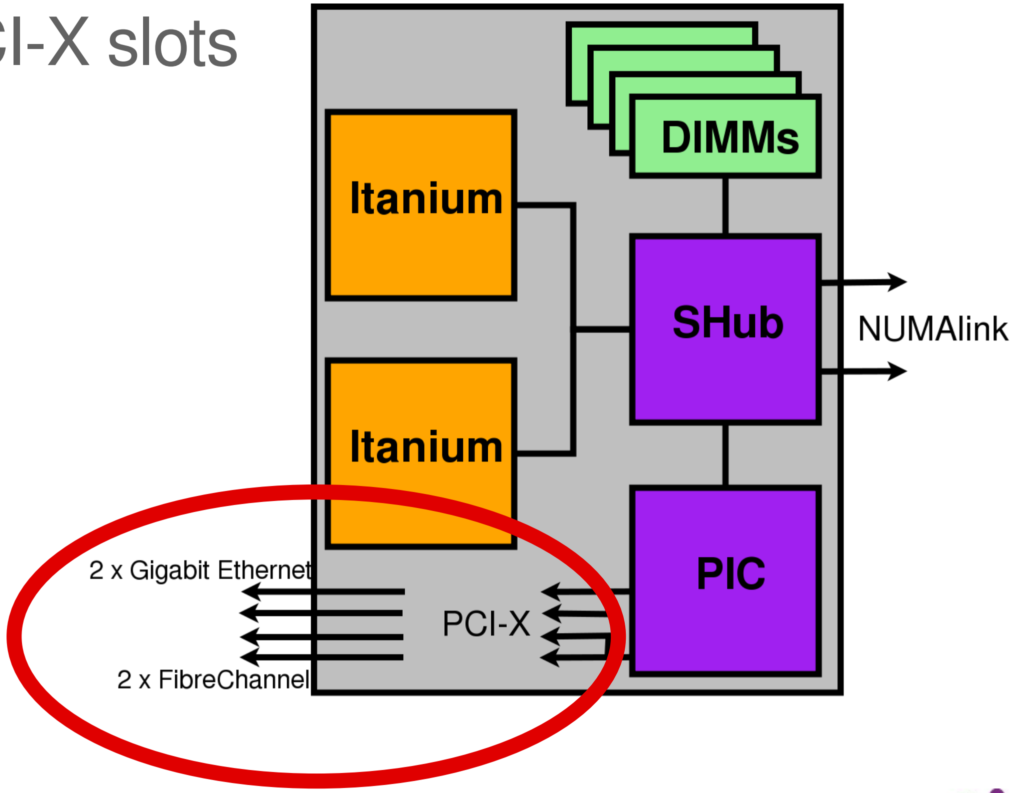
Building Block

- Altix A350 “brick”
- 2 Itanium CPUs
- 12 DIMM slots (4 – 24 GiB)
- lots of memory bandwidth



Building Block

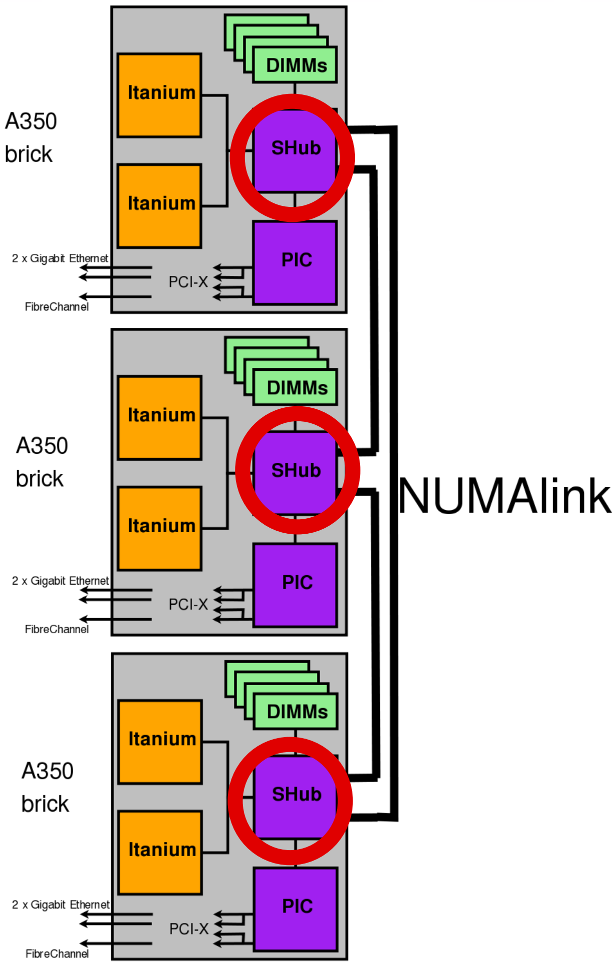
- 4 x 64bit 133 MHz PCI-X slots
- 2 Gigabit Ethernets
- RAID attached
  with FibreChannel



sgi INNOVATION FOR RESULTS

Building A Bigger Server

- Connect multiple bricks with NUMALink™ up to 16 CPUs



sgi INNOVATION FOR RESULTS

NFS Sucks!

- Yeah, we all knew that

NFS Sucks!

- But really, on Altix it **sucked slooooowly**
- 2 x 1.4 GHz McKinley slower than
  2 x 800 MHz MIPS
- 6 x Itanium -> 8 x Itanium
  33% more power, 12% more NFS throughput
- With fixed # clients, more CPUs was slower!
- Simply did not scale; CPU limited

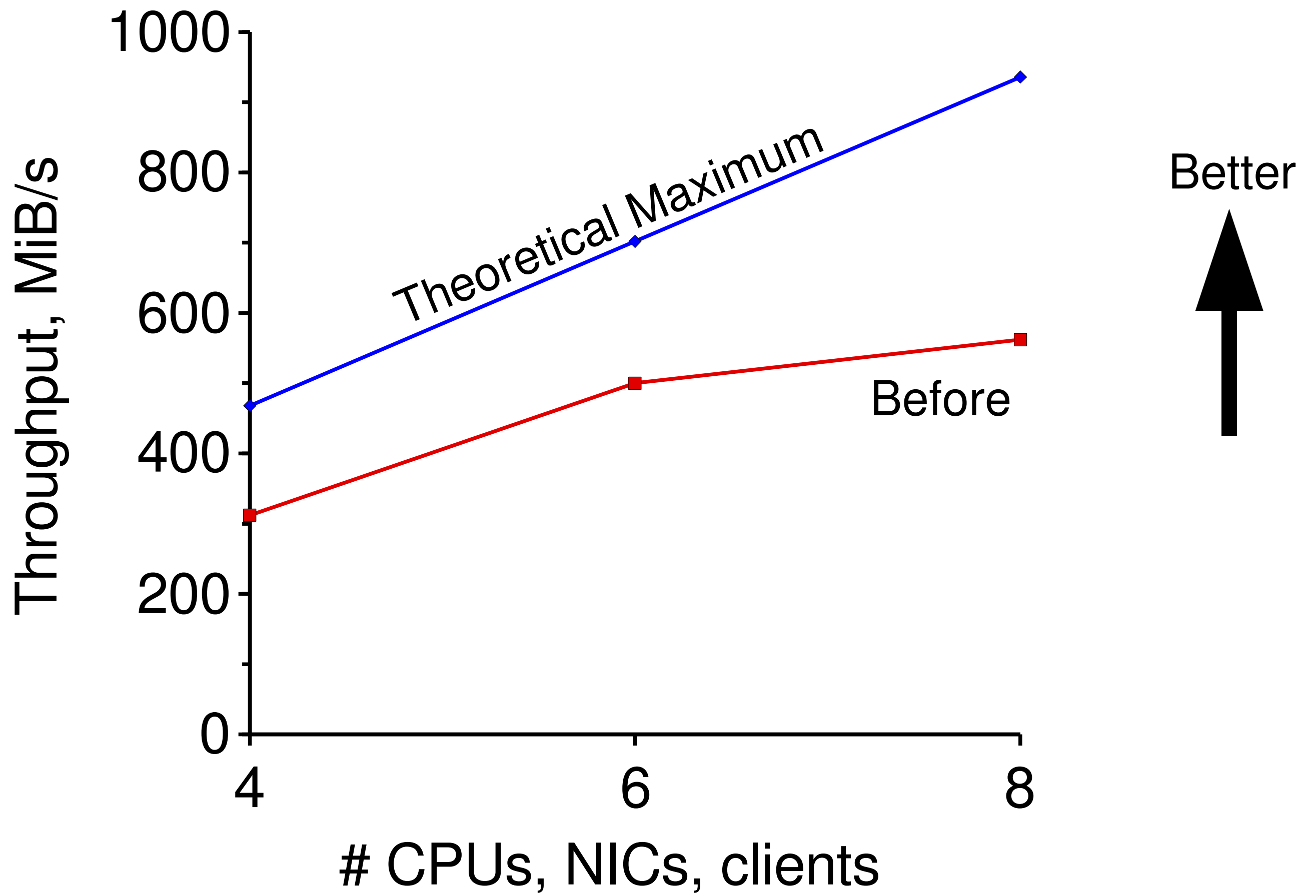
NFS Sucks!

- My mission...

  make the Linux NFS server suck **faster** on NUMA

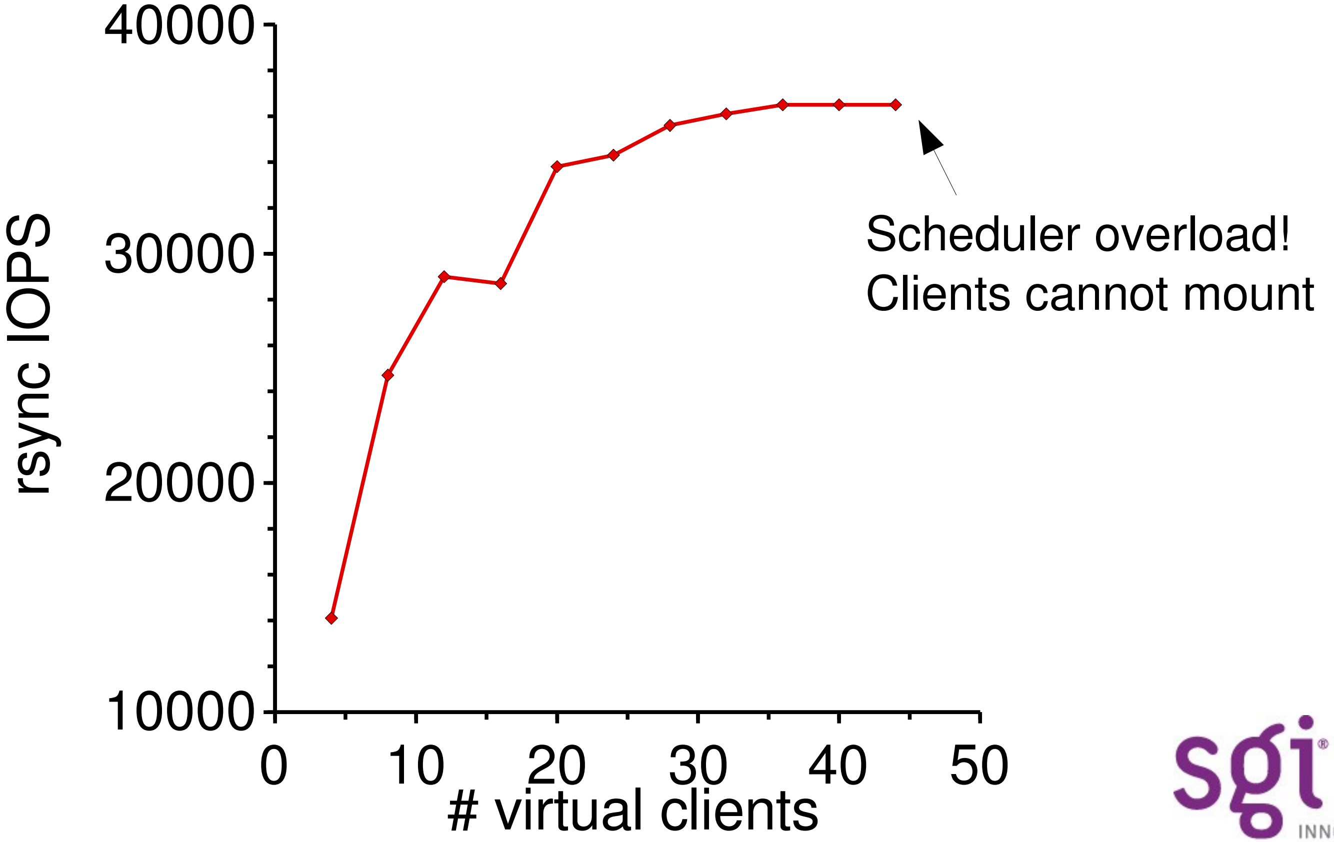
Bandwidth Test

- Throughput for streaming read, TCP, rsize=32K



Call Rate Test

- IOPS for in-memory rsync from simulated Linux 2.4 clients



Overview

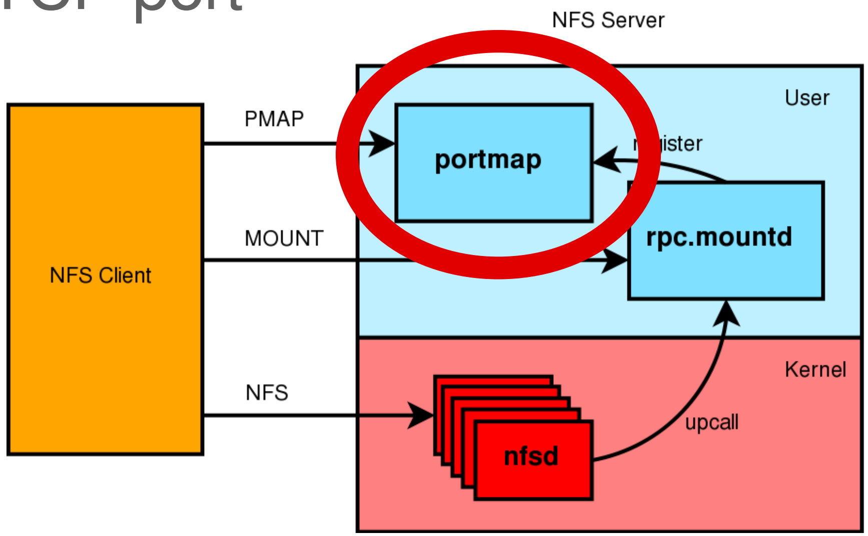
- Introduction
- **Principles of Operation**
- Performance Factors
- Performance Results
- Future Work
- Questions?

INNOVATION FOR RESULTS

Principles of Operation

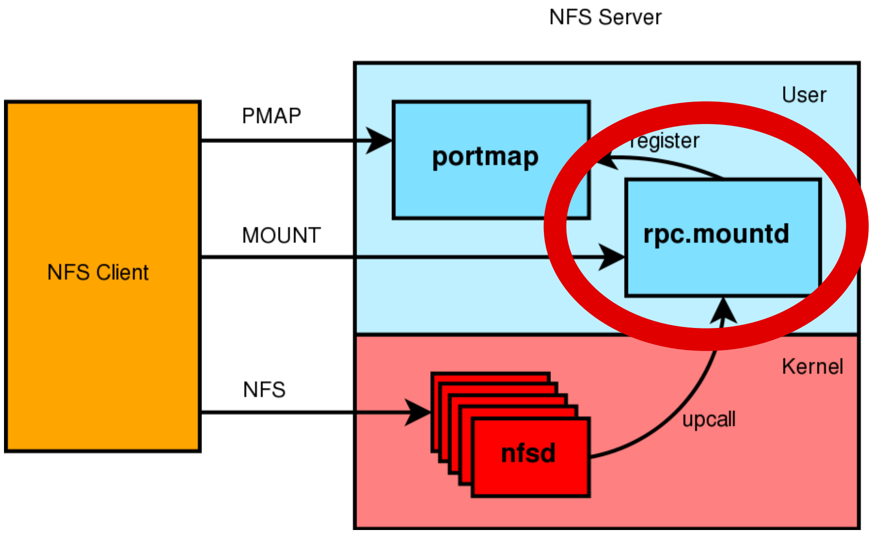
- portmap

  maps RPC program # -> TCP port



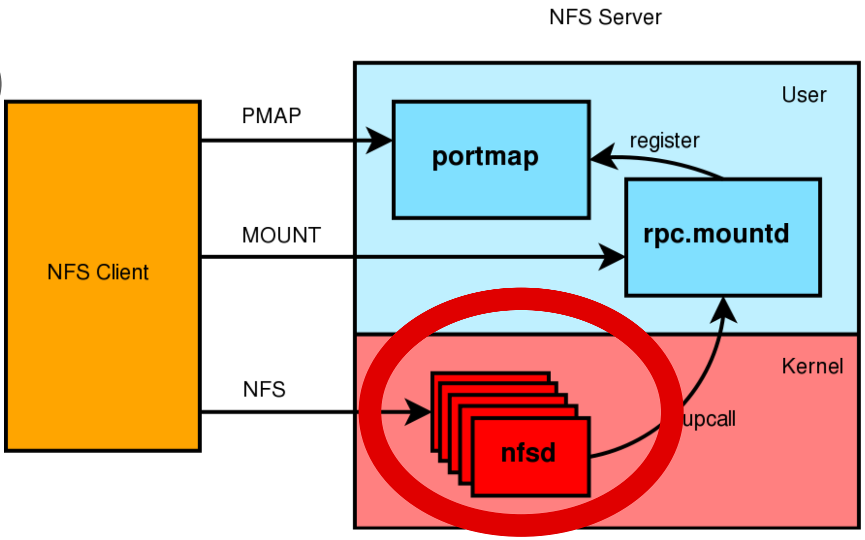
Principles of Operation

- rpc.mountd
- handles MOUNT call
- interprets */etc/exports*



Principles of Operation

- kernel nfsd threads
- global pool
- little per-client state (< v4)
- threads handle calls not clients
- “upcall”s to rpc.mountd



Kernel Data Structures

- *struct svc_socket*
- per UDP or TCP socket

Kernel Data Structures

- *struct svc_serv*
- effectively global
- pending socket list
- available threads list
- permanent sockets list (UDP, TCP rendezvous)
- temporary sockets (TCP connection)

Kernel Data Structures

- *struct ip_map*
- represents a client IP address
- sparse hashtable, populated on demand

Lifetime of an RPC service thread

- **If no socket has pending data, block**
  - normal idle condition

Lifetime of an RPC service thread

- If no socket has pending data, block
  - normal idle condition
- **Take a pending socket from the (global) list**

Lifetime of an RPC service thread

- If no socket has pending data, block
  - normal idle condition
- Take a pending socket from the (global) list
- **Read an RPC call from the socket**

Lifetime of an RPC service thread

- If no socket has pending data, block
  - normal idle condition
- Take a pending socket from the (global) list
- Read an RPC call from the socket
- **Decode the call (protocol specific)**

Lifetime of an RPC service thread

- If no socket has pending data, block
  - normal idle condition
- Take a pending socket from the (global) list
- Read an RPC call from the socket
- Decode the call (protocol specific)
- **Dispatch the call (protocol specific)**
  - actual I/O to fs happens here

Lifetime of an RPC service thread

- If no socket has pending data, block
  - normal idle condition
- Take a pending socket from the (global) list
- Read an RPC call from the socket
- Decode the call (protocol specific)
- Dispatch the call (protocol specific)
  - actual I/O to fs happens here
- **Encode the reply (protocol specific)**

Lifetime of an RPC service thread

- If no socket has pending data, block
  - normal idle condition
- Take a pending socket from the (global) list
- Read an RPC call from the socket
- Decode the call (protocol specific)
- Dispatch the call (protocol specific)
  - actual I/O to fs happens here
- Encode the reply (protocol specific)
- **Send the reply on the socket**

Overview

- Introduction
- Principles of Operation
- **Performance Factors**
- Performance Results
- Future Work
- Questions?

Performance Goals: What is Scaling?

- **Scale workload linearly**
  - from smallest model: 2 CPUs, 2 GigE NICs
  - to largest model: 8 CPUs, 8 GigE NICs

Performance Goals: What is Scaling?

- Scale workload linearly
  - from smallest model: 2 CPUs, 2 GigE NICs
  - to largest model: 8 CPUs, 8 GigE NICs
- **Many clients: Handle 2000 distinct IP addresses**

Performance Goals: What is Scaling?

- Scale workload linearly
  - from smallest model: 2 CPUs, 2 GigE NICs
  - to largest model: 8 CPUs, 8 GigE NICs
- Many clients: Handle 2000 distinct IP addresses
- **Bandwidth: fill those pipes!**

Performance Goals: What is Scaling?

- Scale workload linearly
  - from smallest model: 2 CPUs, 2 GigE NICs
  - to largest model: 8 CPUs, 8 GigE NICs
- Many clients: Handle 2000 distinct IP addresses
- Bandwidth: fill those pipes!
- **Call rate: metadata-intensive workloads**

Lock Contention & Hotspots

- spinlocks contended by multiple CPUs
- oprofile shows time spent in *ia64_spinlock_contention*.

Lock Contention & Hotspots

- on NUMA, don't even need to contend
- cache coherency latency for unowned cachelines
- off-node latency much worse than local
- “cacheline ping-pong”

Lock Contention & Hotspots

- affects data structures as well as locks
- kernel profile shows time spent in un-obvious places in functions
- lots of cross-node traffic in hardware stats

sgi® INNOVATION FOR RESULTS℠

Some Hotspots

- *sv_*lock spinlock *in struct svc_serv*
  – guards global list of pending sockets, list of pending threads
- split off the hot parts into multiple *svc_pool*s
  – one *svc_pool* per NUMA node
  – sockets are attached to a pool for the lifetime of a call
  – moved temp socket aging from main loop to a timer

sgi® INNOVATION FOR RESULTS™

Some Hotspots

- *struct nfsdstats*
  - global structure
- eliminated some of the less useful stats
  - fewer writes to this structure

Some Hotspots

- readahead params cache hash lock
  - global spinlock
  - 1 lookup+insert, 1 modify per READ call
- split hash into 16 buckets, one lock per bucket

Some Hotspots

- duplicate reply cache hash lock
  - global spinlock
  - 1 lookup, 1 insert per non-idempotent call (e.g. WRITE)
- more hash splitting

Some Hotspots

- lock for struct ip_map cache
  - YA global spinlock
- cached *ip_map* pointer in *struct svc_sock* -- for TCP

NUMA Factors: Problem

- Altix; presumably also Opteron, PPC
- CPU scheduler provides poor locality of reference
  - cold CPU caches
  - aggravates hotspots
- ideally, want replies sent from CPUs close to the NIC
  - e.g. the CPU where the NIC's IRQs go

NUMA Factors: Solution

- make RPC threads node-specific using CPU mask
- only wake threads for packets arriving on local NICs
  - assumes bound IRQ semantics
  - and no irqbalanced or equivalent

NUMA Factors: Solution

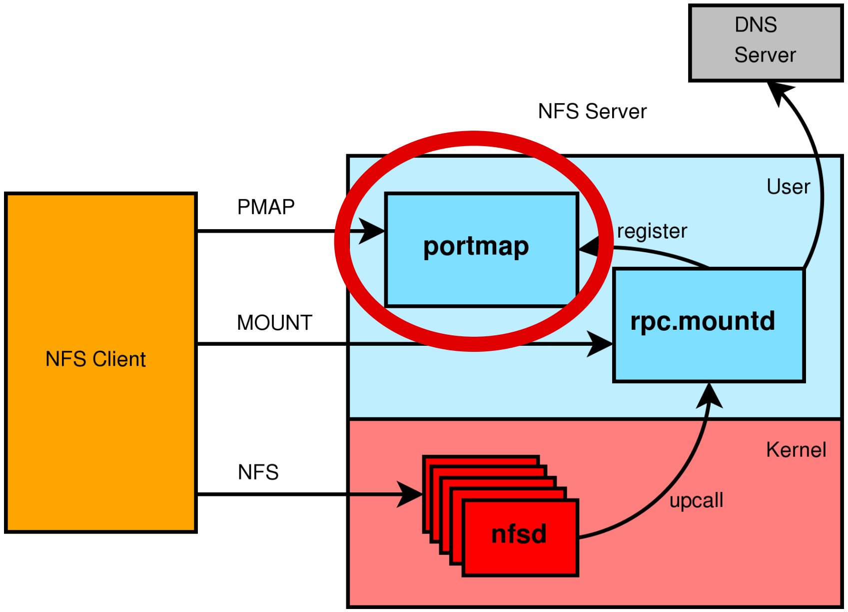
- new file */proc/fs/nfsd/pool_threads*
  - sysadmin may get/set number of threads per pool
  - default round-robins threads around pools

Mountstorm: Problem

- hundreds of clients try to mount in a few seconds
  - e.g. job start on compute cluster
- want parallelism, but Linux serialises mounts 3 ways

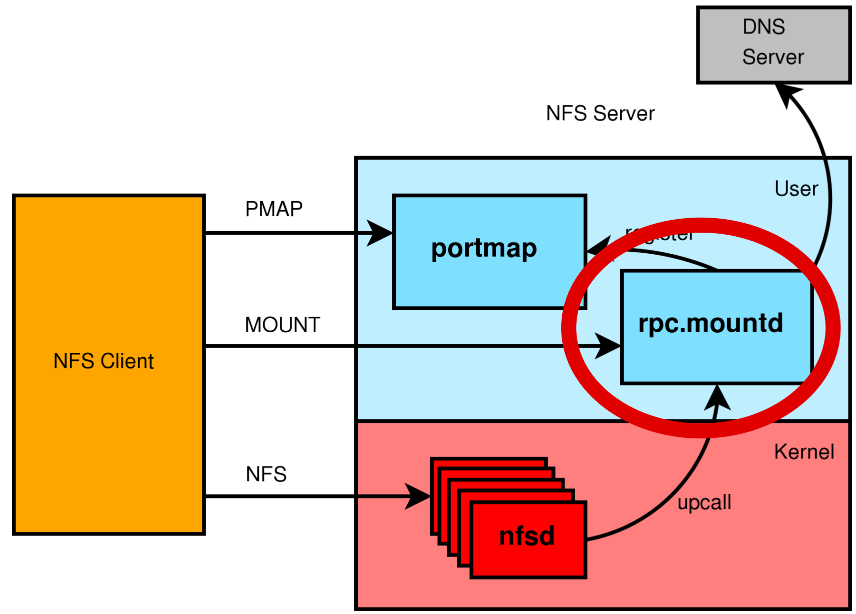
Mountstorm: Problem

- single threaded portmap



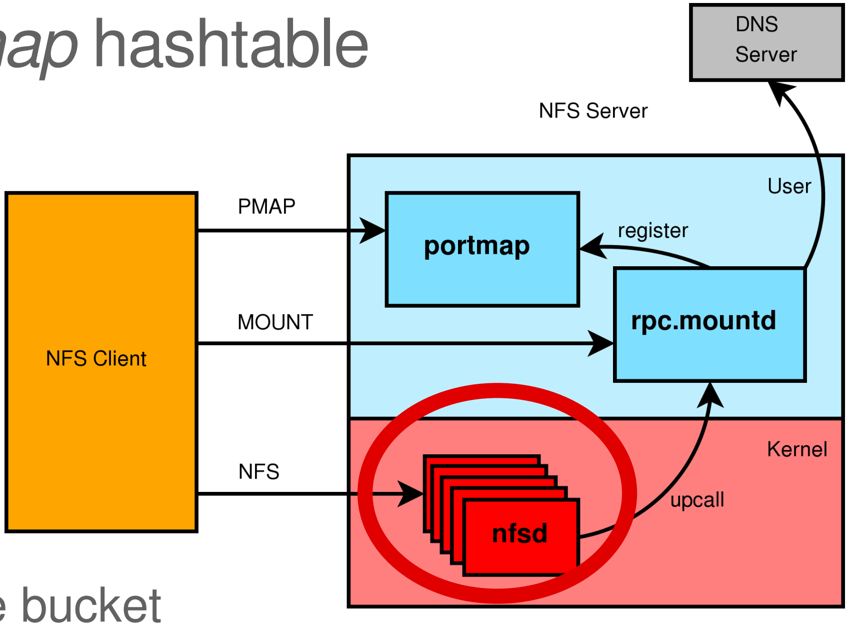
Mountstorm: Problem

- single threaded rpc.mountd
- blocking DNS reverse lookup
- & blocking forward lookup
  - workaround by adding all clients to local */etc/hosts*
- also responds to “upcall” from kernel on 1^{st} NFS call



Mountstorm: Problem

- single-threaded lookup of *ip_map* hashtable
- in kernel, on 1st NFS call from new address
- spinlock held while traversing
- kernel little-endian 64bit IP address hashing balance bug
  - > 99% of *ip_map* hash entries on one bucket



Mountstorm: Problem

- worst case: mounting takes so long that many clients timeout and the job fails.

Mountstorm: Solution

- simple patch fixes hash problem (thanks, iozone)
- combined with hosts workaround:
  can mount 2K clients

Mountstorm: Solution

- multi-threaded rpc.mountd
- surprisingly easy
- modern Linux rpc.mountd keeps state
  - in files and locks access to them, or
  - in kernel
- just fork() some more rpc.mountd processes!
- parallelises hosts lookup
- can mount 2K clients *quickly*

Duplicate reply cache: Problem

- **sidebar**: why have a repcache?
- see Olaf Kirch's OLS2006 paper
- non-idempotent (NI) calls
- call succeeds, reply sent, reply lost in network
- client retries, 2nd attempt fails: bad!

Duplicate reply cache: Problem

- repcache keeps copies of replies to NI calls
- every NI call must search before dispatch, insert after dispatch
- e.g. WRITE
- not useful if lifetime of records < client retry time (typ. 1100 ms).

Duplicate reply cache: Problem

- current implementation has fixed size 1024 entries: supports 930 calls/sec
- we want to scale to ~10^5 calls/sec
- so size is 2 orders of magnitude too small
- NFS/TCP rarely suffers from dups
- yet the lock is a global contention point

Duplicate reply cache: Solution

- modernise the repcache!
- automatic expansion of cache records under load
- triggered by largest age of a record falling below threshold

Duplicate reply cache: Solution

- applied hash splitting to reduce contention
- tweaked hash algorithm to reduce contention

Duplicate reply cache: Solution

- implemented hash resizing with lazy rehashing...
- for SGI NAS, not worth the complexity
- manual tuning of the hash size sufficient

CPU scheduler overload: Problem

- Denial of Service with high call load (e.g. rsync)

CPU scheduler overload: Problem

- knfsd wakes a thread for every call
- all 128 threads are runnable but only 4 have a CPU
- load average of ~120 eats the last few% in the scheduler
- only kernel nfsd threads ever run

CPU scheduler overload: Problem

- user-space threads don't schedule for...minutes
- portmap, rpc.mountd do not accept() new connections before client TCP timeout
- new clients cannot mount

CPU scheduler overload: Solution

- limit the # of nfsds woken but not yet on CPU

NFS over UDP: Problem

- bandwidth limited to ~145 MB/s no matter how many CPUs or NICs are used
- unlike TCP, a single socket is used for all UDP traffic

NFS over UDP: Problem

- when replying, knfsd uses the socket as a queue for building packets out of a header and some pages.
- while holding svc_socket->sk_sem
- so the UDP socket is a bottleneck

NFS over UDP: Solution

- multiple UDP sockets for receive
- 1 per NIC
- bound to the NIC (standard linux feature)
- allows multiple sockets to share the same port
- but device binding affects routing,
  so can't send on these sockets...

NFS over UDP: Solution

- multiple UDP sockets for send
- 1 per CPU
- socket chosen in NFS reply send path
- new UDP_SENDONLY socket option
- not entered in the UDP port hashtable, cannot receive

Write performance to XFS

- Logic bug in XFS writeback path
  - On write congestion kupdated incorrectly blocks holding i_sem
  - Locks out nfsd
- System can move bits
  - from network
  - or to disk
  - but not both at the same time
- Halves NFS write performance

Tunings

- maximum TCP socket buffer sizes
- affects negotiation of TCP window scaling at connect time
- from then on, knfsd manages its own buffer sizes
- tune 'em up high.

sgi INNOVATION FOR RESULTS

Tunings

- tg3 interrupt coalescing parameters
- bump upwards to reduce softirq CPU usage in driver

Tunings

- VM writeback parameters
- bump down *dirty_background_ratio*, *dirty_writeback_centisecs*
- try to get dirty pages flushed to disk before the COMMIT call
- alleviate effect of COMMIT latency on write throughput

Tunings

- *async* export option
- only for the brave
- can improve write performance...or kill it
- **unsafe!!** data not on stable storage but client thinks it is

Tunings

- *no_subtree_check* export option
- no security impact if you only export mountpoints
- can save nearly 10% CPU cost per-call
- technically more correct NFS fh semantics

Tunings

- Linux' ARP response behaviour suboptimal
- with shared media, client traffic jumps around randomly between links on ARP timeout
- common config when you have lots of NICs
- affects NUMA locality, reduces performance
- /proc/sys/net/ipv4/conf/$eth/arp_ignore
  .../arp_announce

Tunings

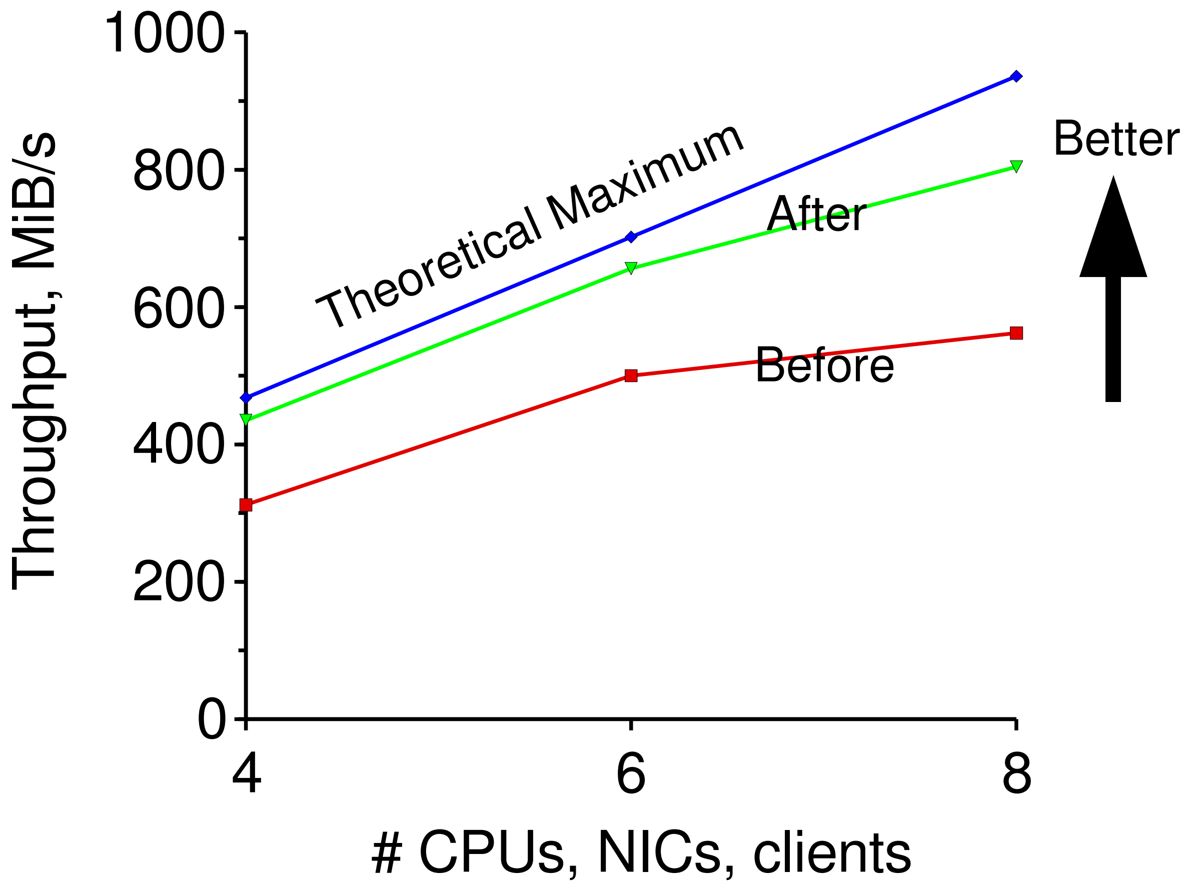
- ARP cache size
- default size overflows with about 1024 clients
- /proc/sys/net/ipv4/neigh/default/gc_thresh3

Overview

- Introduction
- Principles of Operation
- Performance Factors
- **Performance Results**
- Future Work
- Questions?

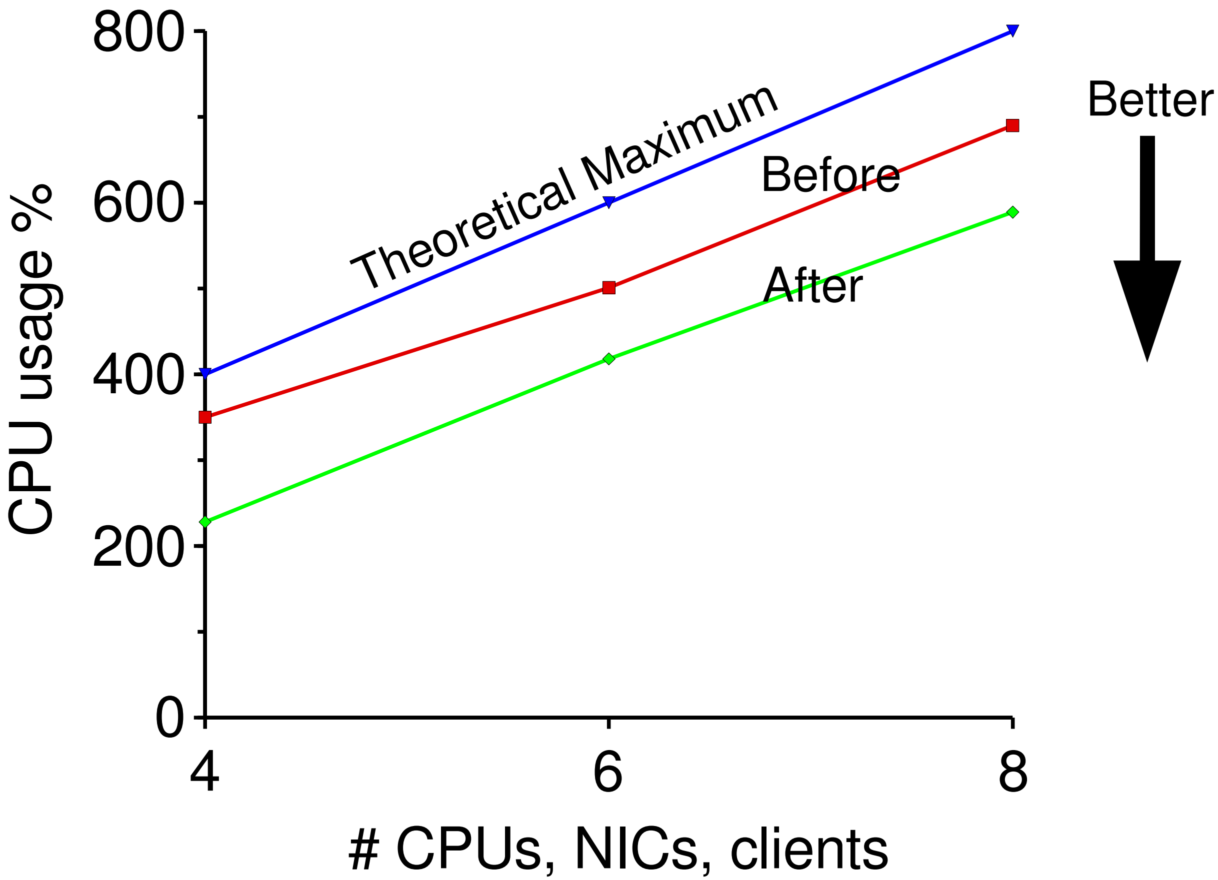
Bandwidth Test

- Throughput for streaming read, TCP, rsize=32K



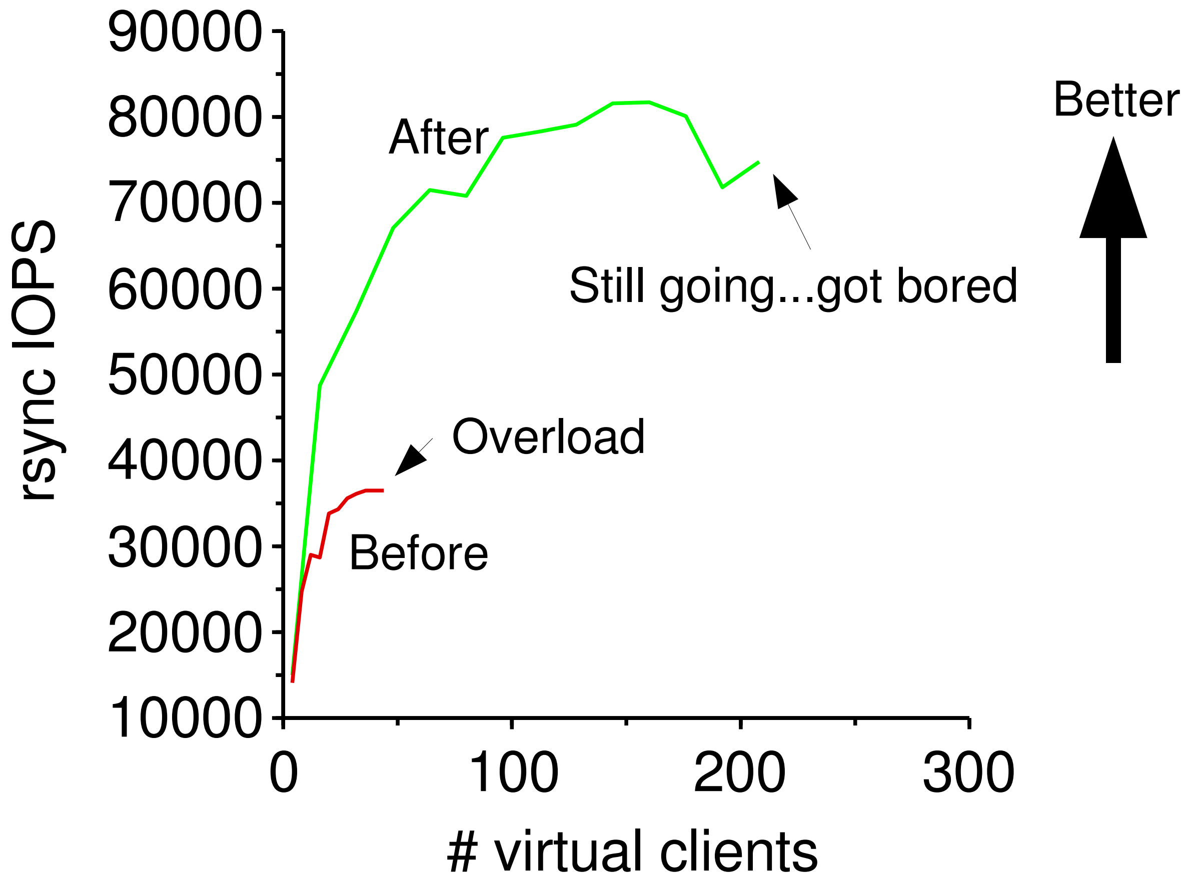
Bandwidth Test: CPU Usage

- %sys+%intr CPU usage for streaming read, TCP, rsize=32K



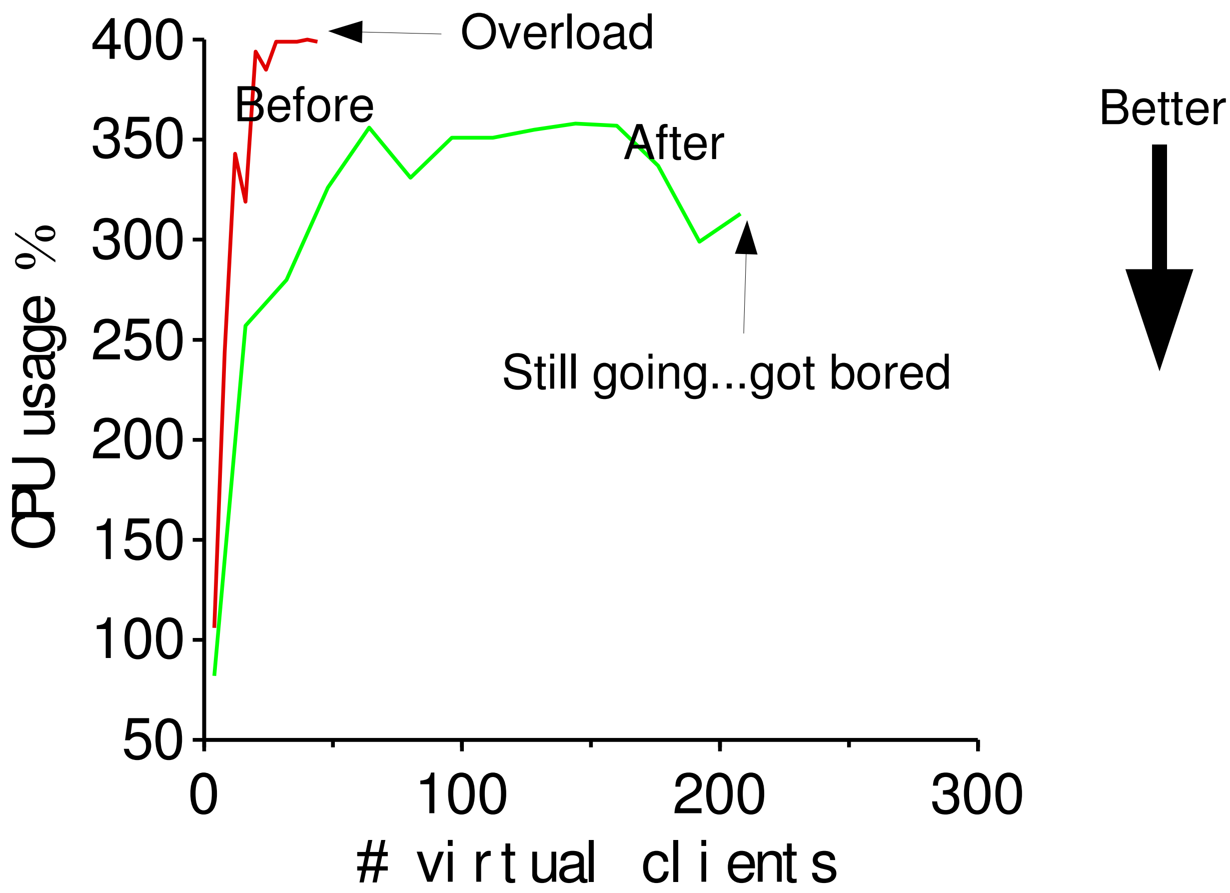
Call Rate Test

- IOPS for in-memory rsync from simulated Linux 2.4 clients, 4 CPUs 4 NICs



Call Rate Test: CPU Usage

- %sys +%intr CPU usage for in-memory rsync from simulated Linux 2.4 clients



Performance Results

- More than doubled SPECsfs result
- Made possible the 1st published Altix SPECsfs result

Performance Results

- **July 2005: SLES9 SP2 test on customer site "W" with 200 clients: failure**

Performance Results

- July 2005: SLES9 SP2 test on customer site "W" with 200 clients: failure
- **May 2006: Enhanced NFS test on customer site "P" with 2000 clients: success**

Performance Results

- July 2005: SLES9 SP2 test on customer site "W" with 200 clients: failure
- May 2006: Enhanced NFS test on customer site "P" with 2000 clients: success
- **Jan 2006: customer “W” again...fingers crossed!**

Overview

- Introduction
- Principles of Operation
- Performance Factors
- Performance Results
- **Future Work**
- Questions?

Read-Ahead Params Cache

- cache of *struct raparm* so NFS files get server-side readahead behaviour
- replace with an *open file cache*
  - avoid *fops->release* on XFS truncating speculative allocation
  - avoid *fops->open* on some filesystems

sgi INNOVATION FOR RESULTS

Read-Ahead Params Cache

- need to make the cache larger
  - we use it for writes as well as reads
  - current sizing policy depends on #threads
- issue of managing new *dentry/vfsmount* references
  - removes all hope of being able to unmount an exported filesystem

One-copy on NFS Write

- NFS writes now require two *memcpy*
  - network *sk_buff* buffers -> nfsd buffer pages
  - nfsd buffer pages -> VM page cache

- the 1st of these can be removed

One-copy on NFS Write

- will remove need for most RPC thread buffering
  - make nfsd memory requirements independent of number of threads
- will require networking support
  - new APIs to extract data from sockets without copies
- will require rewrite of most of the server XDR code

- not a trivial undertaking

Dynamic Thread Management

- number of nfsd threads is a crucial tuning
  - Default (4) is almost always too small
  - Large (128) is wasteful, and can be harmful
- existing advice for tuning is frequently wrong
- no metrics for correctly choosing a value
  - existing stats hard to explain & understand, and ***wrong***

Dynamic Thread Management

- want *automatic* mechanism:
- control loop driven by load metrics
- sets # of threads
- NUMA aware
- manual limits on threads, rates of change

Multi-threaded Portmap

- portmap has read-mostly in-memory database
- not as trivial to MT as rpc.mountd was!
- will help with mountstorm, a little
- code collision with NFS/IPv6 renovation of portmap?

Acknowledgements

- this talk describes work performed at SGI Melbourne, July 2005 – June 2006
  - thanks for letting me do it
  - ...and talk about it.
  - thanks for all the cool toys.

Acknowledgements

- kernel & nfs-utils patches described are being submitted

- thanks to code reviewers
  - Neil Brown, Andrew Morton, Trond Myklebust, Chuck Lever, Christoph Hellwig, J Bruce Fields and others.

sgi® INNOVATION FOR RESULTS

References

- SGI *http://www.sgi.com/storage/*.
- Olaf Kirch, "Why NFS Sucks", *http://www.linuxsymposium.org/2006/linuxsymposium_procv2.pdf*
- PCP *http://oss.sgi.com/projects/pcp*
- Oprofile *http://oprofile.sourceforge.net/*
- fsx *http://www.freebsd.org/cgi/cvsweb.cgi/src/tools/regression/fsx/*
- SPECsfs *http://www.spec.org/sfs97r1/*
- fsstress *http://oss.sgi.com/cgi-bin/cvsweb.cgi/xfs-cmds/xfstests/ltp/*
- TBBT *http://www.eecs.harvard.edu/sos/papers/P149-zhu.pdf*

INNOVATION FOR RESULTS™

Advertisement

- SGI Melbourne is hiring!
  - Are you a Linux kernel engineer?
  - Do you know filesystems or networks?
  - Want to do QA in an exciting environment?
  - Talk to me later

Overview

- Introduction
- Principles of Operation
- Performance Factors
- Performance Results
- Future Work
- **Questions?**