Aalto University
School of Science
and Technology

Flexbuf

Tomi Salminen, Ari Mujunen

Metsähovi Radio Observatory
Aalto University, School of Electrical Engineering
tlsalmin@kurp.hut.fi

October 23, 2012

Introduction on Flexbuff

- A NEXPReS EU-project started in 2011 for "Provisioning High-Bandwidth, High-Capacity Networked Storage on Demand" => A high speed data recorder and streamer
- Uses Commercial-Off-The-Self (COTS) hardware.
  - A modern multicore processor
  - A motherboard with enough PCIE slots to facilitate the hardware
  - Efficient SATA-controllers
  - 10Gb NICS
  - ~20 x 2TB hard drives
  - HW-configuration is only a recommendation.
- Runs vlbi-streamer software (FOSS GPLv3 licensed @ http://code.google.com/p/vlbi-streamer/)

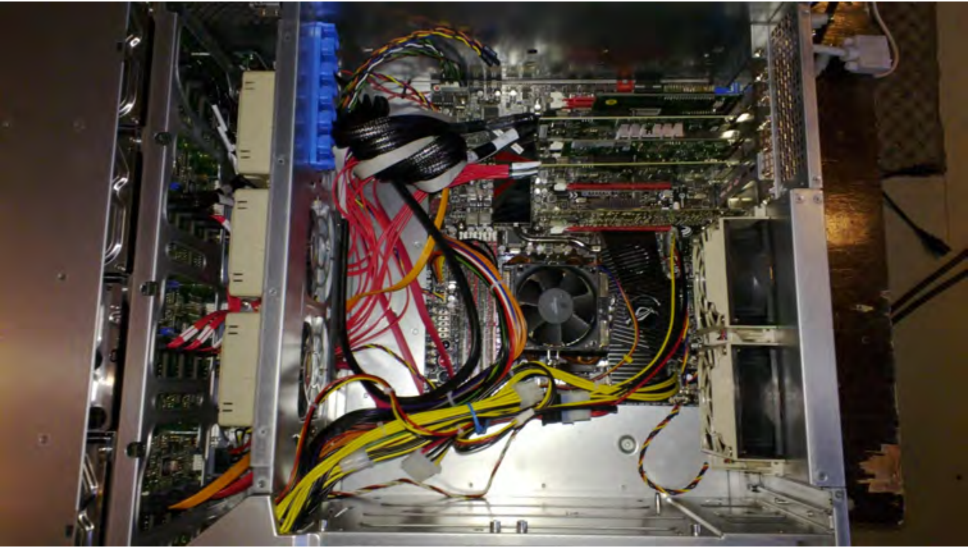
Hardware on Ara

Motherboard	Crosshair IV Extreme AMD 890FX + SB850 chipset
CPU	AMD Phenom ™ II X6 1090T
Memory	16GB 667 Mhz (Motherboard acting up. Target: 1333Mhz)
NIC	Chelsio T320 10GbE Dual Port Adapter
NIC	Intel 82599EB 10-Gigabit SFI/SFP+ Network Connection
SATA	2 x SAS2008 PCI-Express Fusion-MPT SAS-2 [Falcon]
SATA	Internal JMicron controllers
HD	22 x 1-2 TB drives

Hardware on Watt

Motherboard	Supermicro X8DTH
CPU	2 x Intel Xeon E5620 @ 2.4 Ghz
Memory	20GB 1066 Mhz Memory
NIC	Intel 82598EB 10-Gigabit AT CX4 Network Connection
SATA	4 x SAS2008 PCI-Express Fusion-MPT SAS-2 [Falcon]
HD	36 x 2TB drives

Hardware



Challenges

- Ye olde spinning hard drives maximum rate is between 60-120 MB/s
  - Speeds slows as data is written to inner tracks due to data density increasing
  - Seek times very costly. Need sequential operations.
  - More hard drives => More hard drives breaking.
- Above notes hint that writing data shouldn't be done in a rigid architecture
  - Speeds per drive cannot be guaranteed.
  - A faulty hard drive should not be able to slow down nor fail a recording.

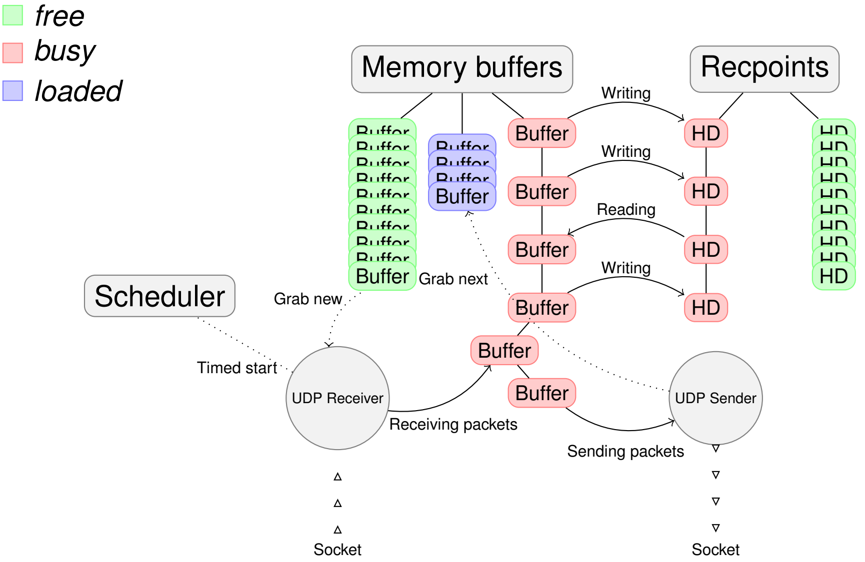
- TCP/IP flow-control ramps speed up too slow, one packet lost will drop speed back to minimum.
  - The FPGA packetizers are not programmed for the TCP-procedure.
  - Problems like buffer bloat in TCP are currently being addressed in kernel development
  - Kernel hides TCP and UDP socket differences almost completely
- Use UDP-packets. Small packet loss is accepted.
  - UDP packet loss in short connections is rare on modern hardware
- 1500 Byte packets @ 10 Gb/s $\approx$ 900k packets per second. NIC interrupt rates can be an issue.
- Without using special drivers the kernel socket buffer needs to be emptied before overflow
  - Receiving thread shouldn't be blocked or too heavy.
- With special drivers somebody needs to keep them updated and working with newer kernel or else..

- Special socket options like rxring and fanout available in newer kernels.
  - Direct copy to memory and splitting of work to multiple threads. Just what we need, except..
  - ..Interrupt rates go up to $\approx$ 80% on 6Gb/s
  - ..Massive packet loss on fast streams.
  - Problems most likely due to dropping interrupt mitigation stuff on driver and the features dev-status.
  - Implemented once for vlbi-streamer testing, but not maintained
- Network connections will keep evolving.
  - A hardware specific solution will be obsolete relatively fast.
  - Requires reimplementation and yet another project.
- Same thing with writing to non-volatile memory
  - SSDs are becoming affordable. A hardware specific solution would be very different

VLBI-streamer

- A hardware independent recorder and streamer.
- Hard drives as a pools of resources, used sequentially when reserved.
- Memory split into file size (256MB-512MB) buffers.
- Memory buffers as threads that handle disk writing/reading
- Receiving thread only fills buffers.
- Modular
  - Disk write-ends can be changed without changes to main program
  - Same with the network side

- Uses a schedule (eg. VLBI session)
  - Single invocation still available: ./configure –enable-daemon=no
- Arbitrary number of receive/send sessions active.
- Real-time sending and delayed sending of recording.
- Software resiliant to hard drive fails. (Only on receive side atm.)
- Packet resequencing framework.
- Easy to use (Please help by testing it and giving feedback)

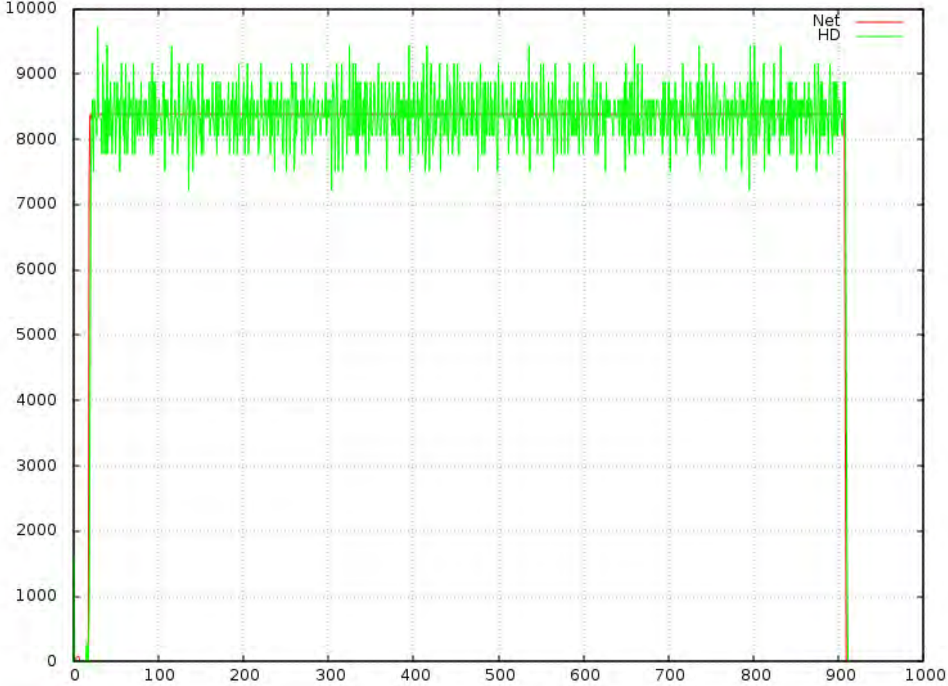

free
busy
loaded
Memory buffers
Recpoints
Buffer
HD
Writing
Reading
Grab next
Grab new
Scheduler
Timed start
UDP Receiver
Receiving packets
UDP Sender
Sending packets
Socket
Socket

Uses

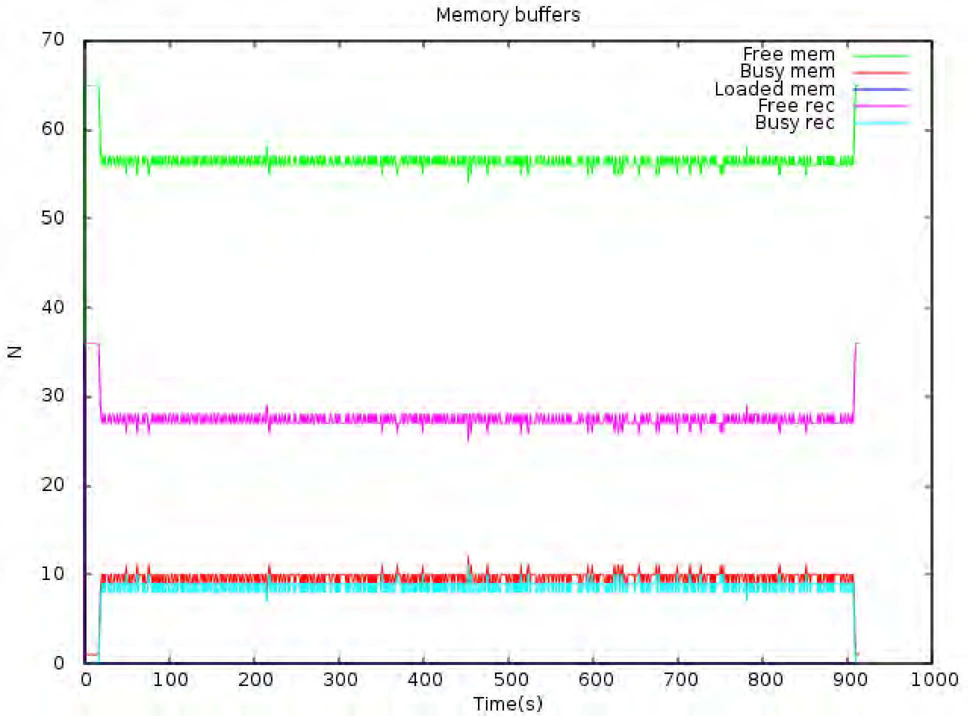
- Main design purpose is station-side buffering
  - Eg. record @ 8Gbps from antenna and stream @ 1Gbps to correlator.
- Long time storage. Set rec points on RAID arrays for redundancy.
- Correlator buffer.
  - Receives multiple streams from stations for correlation
- LOFAR buffers.

Tests

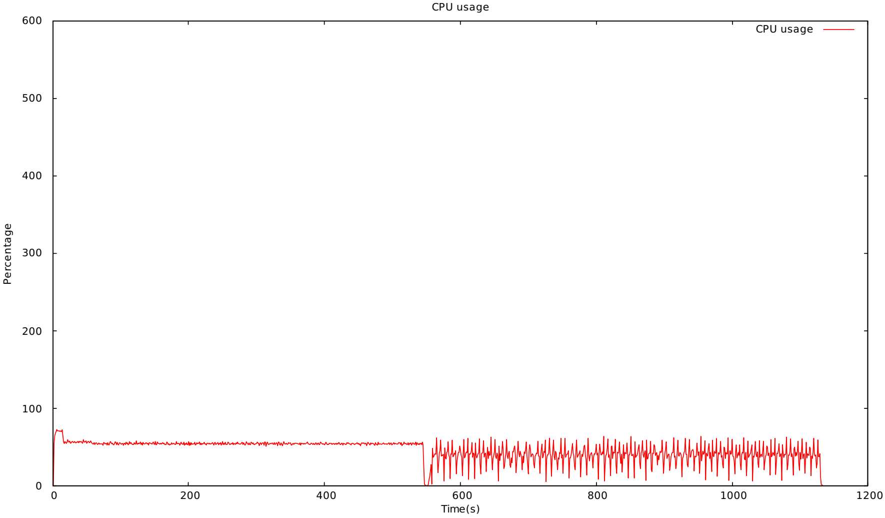
- Long network tests show high speed with no packet loss
  - 12h recording test with 8Gb/s completed without packet loss on multiple sites.
  - Utilizes only a small amount of resources
    - CPU clocks to spare (~80% usage on 6 core system with max 600%
    - Less than half of 22 disks in use. (More than enough to stream old recordings simultaneously)
- Offline throughput tests show architecture working close to HW-limit. Tests without network side:
  - ~32.9 Gb/s with 32 disks on a Xeon.
  - ~17 Gb/s with 22 disks on a AMD Phenom II X6 system.
  - Offline here means: syscall recv commented out, make system think its continuosly receiving packets
  - Note that this also doesn't include copy from kernel socket receive buffer to memory.

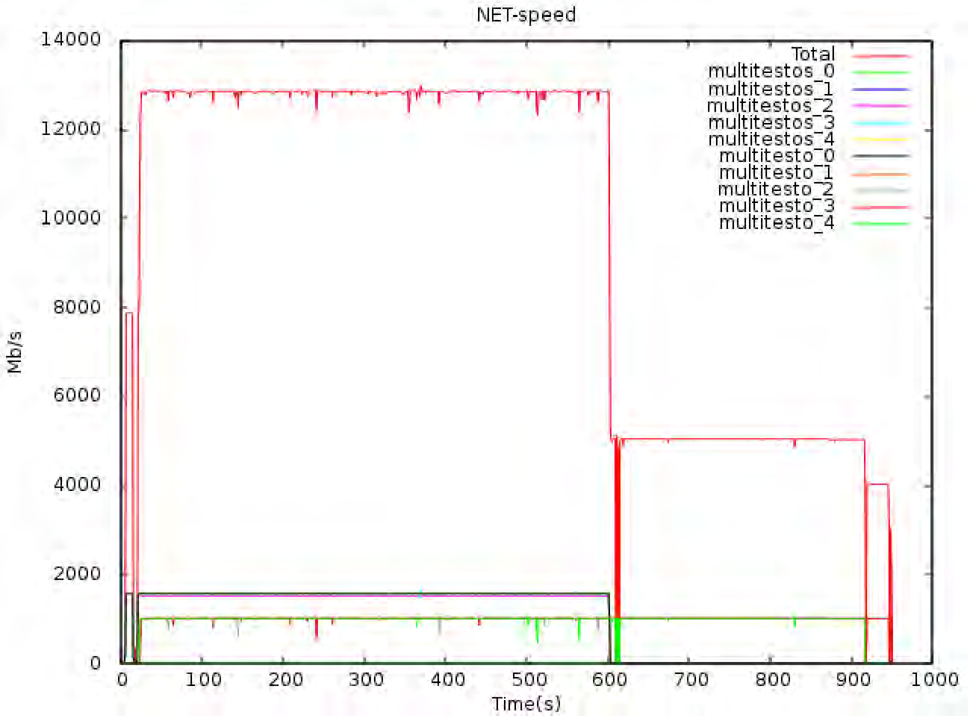


Net
HD
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
100
200
300
400
500
600
700
800
900
1000

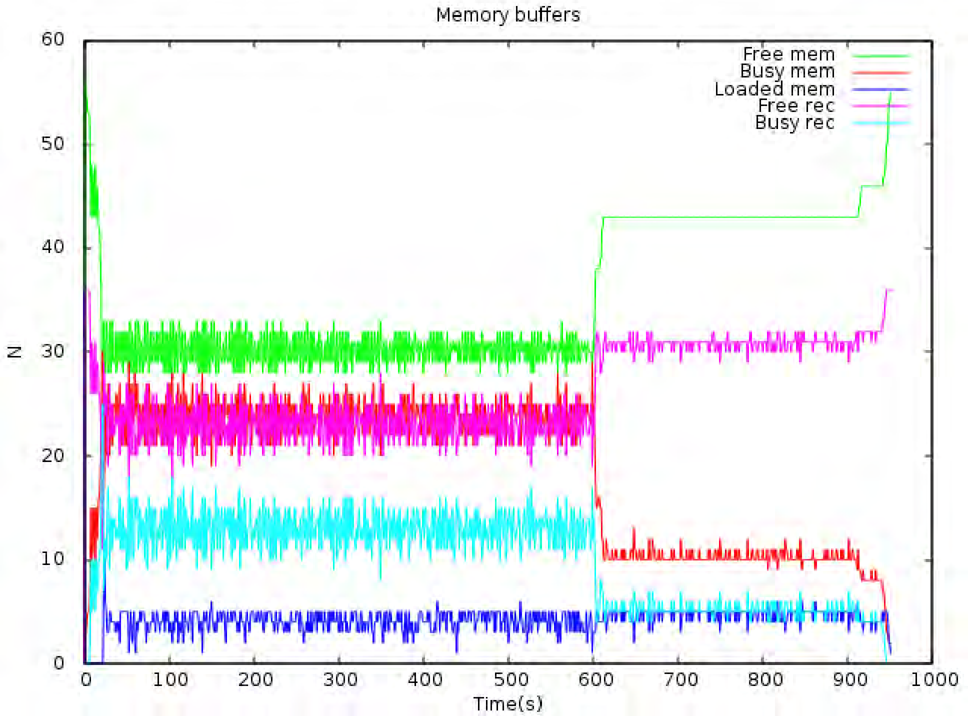


Memory buffers
Free mem
Busy mem
Loaded mem
Free rec
Busy rec
N
Time(s)
0
10
20
30
40
50
60
70
0
100
200
300
400
500
600
700
800
900
1000

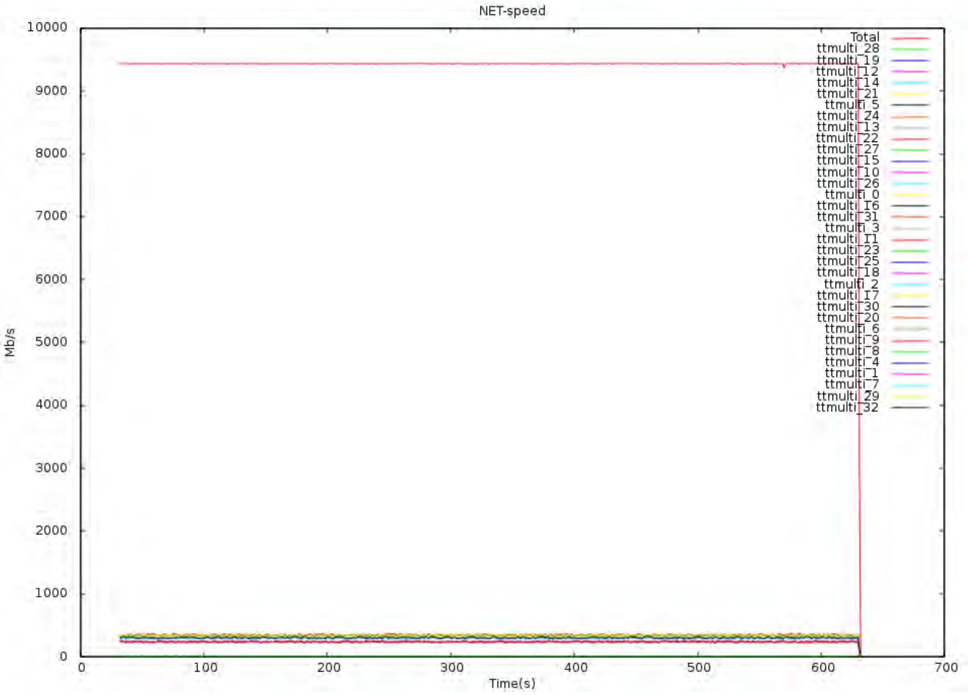

CPU usage
CPU usage
Percentage
Time(s)
0
100
200
300
400
500
600
0
200
400
600
800
1000
1200

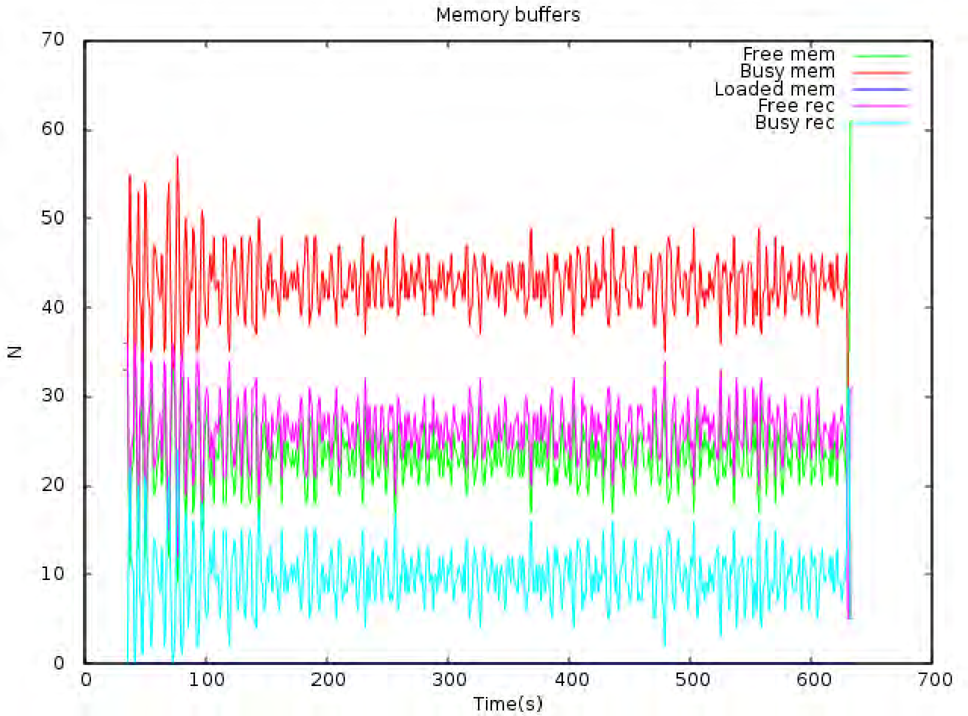


NET-speed
14000
12000
10000
8000
6000
4000
2000
0
Mb/s
0
100
200
300
400
500
600
700
800
900
1000
Time(s)
Total
multitestos_0
multitestos_1
multitestos_2
multitestos_3
multitestos_4
multitesto_0
multitesto_1
multitesto_2
multitesto_3
multitesto_4



Memory buffers
Free mem
Busy mem
Loaded mem
Free rec
Busy rec
N
Time(s)
0
10
20
30
40
50
60
0
100
200
300
400
500
600
700
800
900
1000


NET-speed
Mb/s
Time(s)
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
100
200
300
400
500
600
700
Total
ttmulti_28
ttmulti_19
ttmulti_12
ttmulti_14
ttmulti_21
ttmulti_5
ttmulti_24
ttmulti_13
ttmulti_22
ttmulti_27
ttmulti_15
ttmulti_10
ttmulti_26
ttmulti_0
ttmulti_16
ttmulti_31
ttmulti_3
ttmulti_11
ttmulti_23
ttmulti_25
ttmulti_18
ttmulti_2
ttmulti_17
ttmulti_30
ttmulti_20
ttmulti_6
ttmulti_9
ttmulti_8
ttmulti_4
ttmulti_1
ttmulti_7
ttmulti_29
ttmulti_32


Memory buffers
Free mem
Busy mem
Loaded mem
Free rec
Busy rec
N
Time(s)
0
10
20
30
40
50
60
70
100
200
300
400
500
600
700

Utilities

- Service (aka. Daemon) scripts: Ready after boot.
- Recpoint formatting/tuning/mounting script.
- Hugepage initialization
- Parsing of logs to gnuplottable format
- Plotting of parsed logs. (Daily integration tests for regression testing etc.)
- Queueing scripts.
  - vbs_record nameoftest 300 -s 47338 -q vdif
- Extraction of schedule from snp-file.
- vbs_delete, vbs_ls..

Development and faults

- Delayed sending (eg. send packet every 35 ms) is currently done in busyloop
  - Regular kernel min. sleep times tend to be ~50ms.
  - Multiple sending threads clog system with busy loops.
  - TCP congestion control etc. actually handle this automatically.
    - Needs QoS

Solution 1. An interrupt facility with function pointers.

- Accuracy to be tested.

Solution 2. Kernel socket option for rate control

Solution 3. Kernel Pre-emption and nanosleep (Thanks Paul)

- Set sender threads to higher priorities
- Kernel pre-emption needs to be set => Kernel tuning
- ./configure –enable-nanosleep=yes

Development and faults

- Packet size change requires /etc/init.d/vbs_daemon restart
  - DIRECT I/O requires 4096 byte alignment
  - Buffer division for write granuality
    - $\frac{512MB}{60MB/s} \approx 8.5s$ reservation for a hard drive.
    - Nasty for simultaneous receive and send, since send might just require that specific hard drive.
    - Blocking and priorities on the TODO-list.
  - Granuality degrades large raid performance
    - Write becomes small per disk.
    - Accesses are fast, so no granuality needed.
  - Packet size alignment
    - Filling takes max overhead $\frac{9000-1500}{256*1024*1024} \approx 0.003\%$

Solution Find divisions for all near some (eg 512) MB spot and reserve max per buffer.