MÁSTER EN COMPUTACIÓN DE ALTAS PRESTACIONES
PROYECTO FIN DE MÁSTER

Despregue de servizos HPC en contornos cloud multipropósito

Autor : David Rodríguez Penas
Tutor : Tomás Fernández Pena
Director : Natalia Costas Lago
Director : Ignacio López Cabido

Santiago de Compostela, 1 de Setiembre de 2012

Índice de contenidos

1. Resumen ejecutivo 1
2. Introducción 3
3. Planificación y requisitos del estudio 5
4. Análisis de tecnologías de comunicaciones 7
4.1 Conmutadores 7
4.2 Aplicaciones y protocolos de comunicaciones 13
4.3 Tecnologías de virtualización 17
4.4 Linux y los dispositivos de red 19
4.5 Calidad de servicio 21
5 Estudio de estado del arte 27
6 Propuesta de optimizaciones 37
6.1 Conmutadores 37
6.2 Aplicaciones y protocolos de comunicaciones 38
6.3 Tecnologías de virtualización 38
6.4 Linux y los dispositivos de red 39
6.5 Calidad de servicio 40
7 Piloto 41
7.1 Propuesta de análisis 41
7.2 Diseño y metodología del experimento 42
7.3 Material, despliegue y configuración 43
7.4 Análisis de los resultados del piloto 54
8 Conclusiones y líneas futuras 55
8.1 Conclusiones 55
8.2 Líneas futuras 56
9 Bibliografía 59
Anexo 63

Índice de figuras

Ilustración 1 - Diagrama de Gantt 6
Ilustración 2 - Arquitectura títipa de un conmutador 7
Ilustración 3 - Tipo de procesado de entrada en los conmutadores 8
Ilustración 4 - Implementaciones de colas en los búferes de los conmutadores 11
Ilustración 5 - Ejemplo típico de contención de salida 11
Ilustración 6 - Arquitectura EX 4200 12
Ilustración 7 - Memoria compartida existente en cada PFE del Juniper EX 4200 13
Ilustración 8 - Ejemplo de comunicación en la pila TCP/IP para programas MPI 14
Ilustración 9 - Variaciones de tamaño de la ventana de congestión en TCP 15
Ilustración 10 - Comunicación entre los domU y los dom0 para el acceso a los dispositivos hardware 18
Ilustración 11 - Anillo de memoria compartida utilizada para la comunicación entre los domU y los dom0. 19
Ilustración 12 - Envío y recepción de paquetes en una NIC Gigabit Ethernet 19
Ilustración 13 - Disciplinas de colas FIFO y PFIFO FAST 23
Ilustración 14 - Disciplinas de colas SFQ y TBF 24
Ilustración 15 - Flujo de un paquete/trama en el conmutador 25
Ilustración 16 - Colas de conmutador según tipo de tráfico (clasificación) 25
Ilustración 17 - Esquema de la arquitectura HULL 27
Ilustración 18 - Mecanismo de QoS de Infiniband 29
Ilustración 19 - Solución propuesta por EyeQ 30
Ilustración 20 - Tecnología Smart-Buffer: gestión de buffers dinámica y centralizada 32
Ilustración 21 - Comunicaciones entre domU en arquitecturas para-virtuales 33
Ilustración 22 - Planificador Network I/O Credit Scheduler 35
Ilustración 23 - Configuración de XEN mediante la creación de un Driver Domain 39
Ilustración 24 - Esquema piloto propuesto 41
Ilustración 25 - Test de Intel IMB MPI PingPong y PingPing 43
Ilustración 26 - Latencia inicial de los domU y los dom0 sin contienda 44
Ilustración 27 - Throughput inicial de los domU y dom0 sin contienda 44
Ilustración 28 - Problemas iniciales entre el tráfico MPI y una interferencia UDP de 200Mbps 45
Ilustración 29 - Pingpong con interrupciones UDP de 200 Mbps, tras deshabilitar coalescing 46
Ilustración 30 - Latencias ante las optimizaciones iniciales 47
Ilustración 31 - Ancho de banda ante las optimizaciones iniciales 47
Ilustración 32 - Latencias para PingPong con QoS en conmutador 48
Ilustración 33 - Latencias para PingPing con QoS en conmutador 48
Ilustración 34 - Ancho de banda para PingPing y PingPong con QoS en conmutador 49
Ilustración 35 - Latencias para PingPong con QoS en NIC y conmutador, bajo congestión por UDP. 50
Ilustración 36 - Latencias para PingPing con QoS en NIC y conmutados, bajo congestión por UDP 50
Ilustración 37 - Ancho de banda para PingPong con QoS en NIC y conmutador, bajo congestión por UDP. 51
Ilustración 38 - Ancho de banda para PingPing con QoS en NIC y conmutador, bajo congestión por UDP. 51
Ilustración 39 - Latencia para Pingpong con QoS en NIC y conmutador, bajo congestión por TCP 52
Ilustración 40 - Latencia para PingPing con QoS en NIC y conmutador, bajo congestión por TCP 52
Ilustración 41 - Ancho de banda para PingPong con QoS en NIC y conmutador, bajo congestión por TCP 53
Ilustración 42 - Ancho de banda para PingPing con QoS en NIC y conmutador, bajo congestión por TCP. 53

Índice de tablas

Tabla 1 – Requisitos a cumplir 5
Tabla 2 - Tipos de matrices de conmutación (*switch fabric*) 9

1. Resumen ejecutivo

En el presente proyecto de fin de máster se ha analizado la problemática del despliegue de aplicaciones HPC (*High Performance Computing*) en una infraestructura *cloud* multipropósito (no exclusivamente dedicadas a aplicaciones de esta índole).

Esta iniciativa viene motivada por el elevado coste en el mantenimiento y adquisición de una infraestructura HPC dedicada a computación de altas prestaciones, así como a la posibilidad de aprovechamiento de recursos disponibles en entornos *cloud* de propósito general cuya existencia es cada vez más habitual en las empresas e instituciones.

La implantación de un servicio HPC sobre una plataforma *cloud* genérica y multipropósito, ofrece reducción en costes de infraestructura. Pero no es una tarea sencilla: la competencia de los diferentes tráficos por los recursos de la arquitectura que componen el *cloud* degradará la calidad de servicio de las diferentes tareas HPC.

Para el desarrollo del mismo han acometido las siguientes tareas:

En primer lugar se ha realizado una revisión de todos los elementos del *cloud* multipropósito (dispositivos físicos, protocolos y aplicaciones *software*) donde puede existir competencia entre los diferentes tráficos existentes.

En segundo lugar se ha analizado exhaustivamente el estado del arte en los ámbitos de ultra baja latencia en las comunicaciones en LAN, virtualización y calidad de servicio.

En tercer lugar se han propuesto diversas mejoras en todos los ámbitos analizados encaminadas a la reducción de latencia y mejora del comportamiento del tráfico HPC en convivencia con otro tipo de flujos.

A continuación se ha desarrollado un pequeño piloto en el que se han puesto en práctica algunos de los mecanismos que contribuyen a la convivencia de aplicaciones en este tipo de entornos heterogéneos obteniéndose mejoras en el comportamiento de tráficos HPC y no HPC en convivencia.

A partir tanto de los resultados de las pruebas y como de la diferente literatura estudiada, se proponen los principales puntos de la infraestructura donde existirá contención entre los diferentes tráficos, y posibles soluciones a dichos problemas.

Finalmente se desarrollan las conclusiones del proyecto entre las que cabe destacar:

- Para la implantación de HPC sobre infraestructuras *cloud* multipropósito es necesario un cuidadoso diseño de la infraestructura de red teniendo en cuenta: conmutadores, mecanismos de virtualización, arquitecturas de tarjetas de red así como el análisis de los diferentes flujos de tráfico.
- En muchos casos los diseños y usos tradicionales de red nos llevan a tratar de obtener la máxima utilización del ancho de banda de las líneas disponibles. Sin embargo, cuando se trata de minimizar la latencia eso puede ser contraproducente, porque deriva en la maximización de la ocupación de los búferes, agregando por tanto latencias adicionales.
- La aplicación de mecanismos de calidad de servicio tradicionales es ventajosa y económica, ya que está ampliamente adoptada en conmutadores y sistemas operativos. Durante las pruebas del piloto se identificaron puntos donde no fue

posible la aplicación de medidas de QoS como son las colas de entrada de los conmutadores y de las tarjetas de red, sin descartar que existan otros conmutadores en el mercado que sí las implementen o implementaciones de QoS en los servidores en las cuales esto sea posible.

- Es posible la obtención de mejoras para el tráfico HPC haciendo uso de mecanismos existentes en la actualidad en los diversos sistemas.

2. Introducción

En la actualidad la utilización de tecnologías de *cloud* para realizar computación de altas prestaciones – HPC (*High Performance Computing*) – es muy costosa en recursos, ya que para ofrecer estos servicios con buena calidad es necesario hacerlo de manera dedicada. Estos servicios HPC tienen requisitos exigentes, en latencia de red, gran ancho de banda en las comunicaciones, uso intensivo de CPU, etc; por ello la infraestructura recomendada para esta clase de trabajos tiene un coste elevado y, sin embargo, no se garantiza que los recursos estén continuamente utilizándose.

La implantación de un servicio de computación de alto rendimiento sobre una plataforma de *cloud* genérica multipropósito, es decir que este no esté exclusivamente dedicado a trabajos de esta índole, ofreciendo reducción en costes de infraestructura y optimización en el uso de ésta, no es una tarea sencilla: la competencia de los diferentes tráficos por los recursos de la arquitectura que componen el *cloud* degradará la calidad de servicio de las diferentes tareas HPC.

El avance tecnológico de las técnicas de virtualización de recursos permite que en la actualidad el rendimiento asociado a diferentes parámetros computacionales[1] y de red (FLOPs, latencias, ancho de banda, etc.) comience a estar aproximándose a los que se obtendrían con una infraestructura física. Aun así, existe el problema de que la virtualización introduce cierto *overhead*.

Las infraestructuras de computación de alto rendimiento utilizan redes de interconexión caras y sofisticadas, como Infiniband o Myrinet, de mayor coste y prestaciones, en relación a comportamiento determinístico, que pueden ofrecer redes Ethernet genéricas de 1 gigabit comúnmente utilizadas en los *clouds* privados.

Existen además elementos propios de las redes de propósito general, como es el protocolo TCP, ampliamente utilizado en éstos como mecanismo desarrollado para ofrecer un transporte fiable de datos, cuyo uso puede degradar el rendimiento en entornos HPC debido al procesamiento de sus largas cabeceras en los paquetes junto a sus mecanismos de control de congestión siendo causante de problemas de contención en la infraestructura de red.

Este estudio tiene el objetivo de analizar los posibles puntos de contención de tráfico que pueden provocar degradación del rendimiento de las aplicaciones HPC en plataformas de uso compartido.

En los siguientes capítulos se analizarán las diferentes tecnologías de red involucradas en el despliegue de un cloud multipropósito y las posibles implicaciones de cara al rendimiento de las aplicaciones de HPC: tarjetas de red, gestión de la red virtualizada, conmutadores (analizando el caso específico de los conmutadores Juniper de la serie EX), aplicaciones y protocolos.

[1] Se recomienda consultar otras publicaciones para solventar este tipo de incidencias "Underutilizing resources for HPC on *clouds*" ([15] Iakymchuk, Napper, & Bientinesi, 2010), *"HPC on Competitive Cloud Resources"* ([02] Bientinesi, Iakymchuk, & Napper, 2010).

A continuación se realizará un estudio del estado del arte en las temáticas relevantes al escenario que nos ocupa: ultra baja latencia, virtualización y calidad de servicio.

De la información recabada se elaborará una propuesta de los elementos susceptibles de ser optimizados, de aquellos que pueden generar problemas de rendimiento y de aquellos que son objeto de investigación a día de hoy.

En los siguientes capítulos se muestran los diferentes test de rendimiento realizados a raíz de la implementación de un pequeño piloto de HPC sobre *cloud* y el análisis de los resultados obtenidos en el mismo.

Por último se definirán las conclusiones obtenidas y líneas de trabajo futuras.

3. Planificación y requisitos del estudio

El objetivo de este trabajo es el análisis de la problemática del despliegue de servicios HPC en entornos multipropósito. A continuación se indican los requisitos que ha de cumplir el presente informe técnico:

Requisitos del estudio	
Nº	**Descripción del requisito**
1.	Analizar la problemática existente en las aplicaciones HPC en entorno *cloud* multipropósito
Se realizará un estudio sobre el estado del arte y los problemas vinculados a la ejecución de trabajos de alto rendimiento en infraestructuras ***cloud*** multipropósito. Se analizarán las tecnologías de red tanto en el lado servidor (NICs, protocolos de comunicaciones, virtualización de la red), como en la parte de la red de área local (conmutadores y arquitecturas de red en LAN).	
2.	Estudiar soluciones/optimizaciones que ayuden a proporcionar una buena QoS
Una vez analizada la problemática tecnológica existente, se determinarán posibles mejoras para incrementar el rendimiento de las aplicaciones HPC en este tipo de entornos: mecanismos de calidad de servicio (en la parte servidor y en los conmutadores); mejora en las tecnologías de virtualización; en los protocolos de red; uso de hardware específico para eliminación de problemas de contención en los *switches*…	
3.	Desplegar y configurar un piloto experimental básico donde realizar las pruebas
Se desplegará y configurará un piloto experimental donde dos o más servidores estén conectados en una LAN. Se emplearán tecnologías de virtualización con el objetivo de simular un entorno ***cloud***. Se instalará todo el software necesario en ellas, y se configurará la red para agruparlas, tanto en el conmutador como en el servidor, mediante redes virtuales independientes mediante VLANs.	
4.	Diseñar, realizar y almacenar las pruebas en el piloto experimental
Se seleccionarán *benchmarks* sintéticos adecuados para la medida los parámetros a estudiar. Se diseñarán y realizarán las pruebas pertinentes en el piloto bajo diferentes casos, con el objetivo de recopilar datos experimentales.	
5.	Analizar los resultados de las pruebas y proponer soluciones en base a estos
Se analizarán las diferentes pruebas efectuadas con el objetivo de vincular los diferentes puntos problemáticos del experimento. Se propondrán optimizaciones/soluciones viables en la infraestructura.	
6.	Realizar estudio sobre las conclusiones y trabajo futuro a realizar
Se darán una serie de recomendaciones y puntos a controlar en el contexto de la computación de altas prestaciones en las infraestructuras de ***cloud*** multipropósito. También se establecerán diferentes líneas a seguir para avanzar en el estudio sobre la materia.	

Tabla 1 – Requisitos a cumplir

En cuanto a la planificación, se muestra el diagrama de *Gantt* y el plan de tareas que se ha seguido:

Traballo	Nome
300 horas	**Proyecto fin de master**
50 horas	Análisis del estado del arte de soluciones QoS en Cloud
40 horas	Análisis de problemas de tecnologías de virtualización y red
30 horas	Estudio de tecnologías QoS en infraestructuras Cloud
50 horas	Despliegue y configuración de un piloto experimental básico
30 horas	Diseño, ejecución y almacenamiento de las pruebas a realizar
50 horas	Análisis de las pruebas realizadas
30 horas	Establecer conclusiones y líneas de trabajo futuro
20 horas	Finalización de la memoria
0 horas	Fin de proyecto

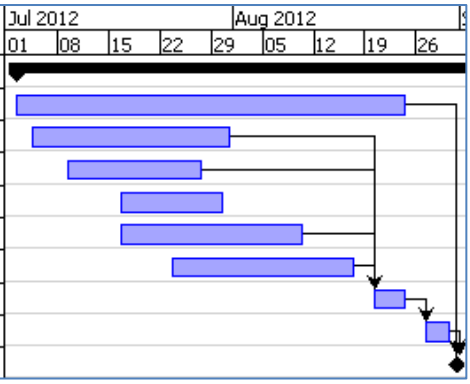

Ilustración 1 – Diagrama de Gantt

- *Análisis de tecnologías y de virtualización y red y análisis de sus posibles puntos de contención de tráfico*: se documentan las diferentes tecnologías y sus problemas de rendimiento, contención y latencias.
- *Análisis del estado del arte de soluciones QoS en cloud multipropósito: se realiza un estudio sobre diferen*tes soluciones a los problemas de compartición de recursos entre tráficos de diferentes orígenes (incluido HPC) en arquitecturas tipo *cloud*.
- *Estudio de tecnologías QoS en infraestructuras cloud*: Se realiza el estudio de diferentes técnicas de calidad de servicio, atendiendo sobre todo a la gestión de los recursos entre diferentes tráficos, priorizando aquellos que sean sensibles a baja latencia/*jitter*.
- *Despliegue y configuración de un piloto experimental básico*: Se monta una pequeña infraestructura física en la que realizar los experimentos.
- *Diseño, ejecución y almacenamiento de las pruebas a realizar*: Se decidirán los parámetros a medir, se ejecutarán las pruebas y se almacenarán los datos para posterior procesado.
- *Análisis de las pruebas realizadas*: Se analizarán los resultados y se indicarán las causas de fallo, proponiendo posibles optimizaciones.
- *Establecer conclusiones, líneas de trabajo futuro*: Se establecerán las conclusiones finales (advertencias de puntos problemáticos) y las recomendaciones para poder solventarlos. Adicionalmente se propondrán líneas de trabajo futuras sobre el contexto de QoS en *clouds* multipropósito.
- *Finalización de memoria:* Se completará y dará formato a la presente memoria, recopilando la documentación generada en las diferentes tareas.

4. Análisis de tecnologías de comunicaciones

Realizaremos inicialmente un estudio de las diferentes tecnologías implicadas en la transmisión de información entre los diferentes *domU* de HPC con el fin de determinar los puntos conflictivos en relación a la compartición de recursos de red y la priorización de su uso. En este capítulo se analizarán las tecnologías de comunicaciones implicadas en las siguientes áreas:

- Conmutadores
- Aplicaciones y protocolos de comunicaciones
- Tecnologías de virtualización y sistemas
- Interfaces de red

4.1 Conmutadores

Los conmutadores son dispositivos digitales que permiten la interconexión de redes de computadores u otros dispositivos. Principalmente trabajan en el nivel 2 del modelo OSI llamado capa de enlace de datos; aunque en la actualidad también tienen funcionalidades para operar en la capa de encaminamiento (nivel 3), aunque en el presente trabajo analizamos únicamente sus funcionalidades a nivel 2.

Su principal cometido es el de comunicar dos o más puntos conmutando datos de un segmento a otro haciendo uso de las direcciones MAC de origen/destino de los dispositivos de la infraestructura.

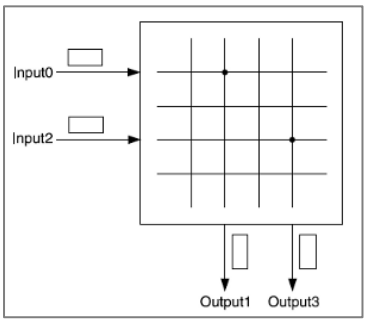

Ilustración 2 - Arquitectura típica de un conmutador

Es importante conocer en detalle cómo funcionan estos dispositivos para analizar la latencia que introducirán en las comunicaciones, ya bien simplemente por pasar a través de ellos, o bien debido a la competencia en recursos internos que existirá entre los diferentes flujos de red.

Para entender cómo funcionan los conmutadores, es conveniente revisar los mecanismos empleados habitualmente dentro de sus arquitecturas. Se indican a continuación los elementos más relevantes extraídos del libro *"High Performance Switches and Routers"* ([05] Chao & Liu, 2007):

- ***Throughput*** y ***speedup:*** El *throughput* o caudal de un conmutador se define como el ratio de la velocidad media de salida agregada por la velocidad media de entrada, cuando todos los puertos de entrada llevan el 100% del tráfico existente en la línea. Será un valor positivo no superior a uno. El *speedup* o aceleración es una medida que expresa cuantas veces es mayor la velocidad interna del conmutador que la velocidad de línea de entrada. Por lo que a una aceleración superior a uno, quiere decir que los buffers estarán siendo usados por los puertos de salida, con lo que se deduce que se transmitirán más de un paquete en el mismo intervalo de tiempo al puerto de salida. Por ello, a mayor aceleración, mayor caudal.

Se podría resumir el funcionamiento de un conmutador en tres fases: la entrada y procesado de las tramas en el dispositivo, el encaminamiento de estos datos a través de la matriz de conmutación (*switch fabric*) y el envío de estos a su destino a través de los puertos de salida.

Por lo tanto, lo primero es recibir y extraer la información del *frame* que llegará a los puertos de entrada desde los dispositivos de origen. Existirán tres formas de hacerlo, dependiendo de si queremos primar la fiabilidad o la rapidez del transporte:

- ***Cut-through mode:*** En este tipo de conmutadores se reciben y examinan solo los primeros 6 bytes de la trama. Estos primeros bytes representan a la dirección MAC destino, lo cual es la información justa y necesaria para tomar una decisión de transmisión. Esto ofrece poca latencia en la transmisión, pero es susceptible a tener problemas de colisión.
- ***Fragment-free mode:*** En este caso reciben y examinan los primeros 64 bytes de la trama, debido a que es donde se pueden detectar las posibles colisiones *ethernet*.
- ***Store-and-forward mode:*** Reciben y examinan la trama completa.

Para casos donde la latencia no es tan crítica se dice que los conmutadores pueden no necesitar implementar las dos primeras opciones, debido a que contienen procesadores rápidos y ASIC (*application-specific integrated circuits*) suficientemente rápidos agregando poco retardo a la comunicación. En el contexto de HPC, en caso de poder elegir, sería conveniente utilizar por lo menos la tecnología *cut-through mode*, debido a que se presupone que en una red local la probabilidad de errores en las tramas es mucho menor que en otros entornos de red.

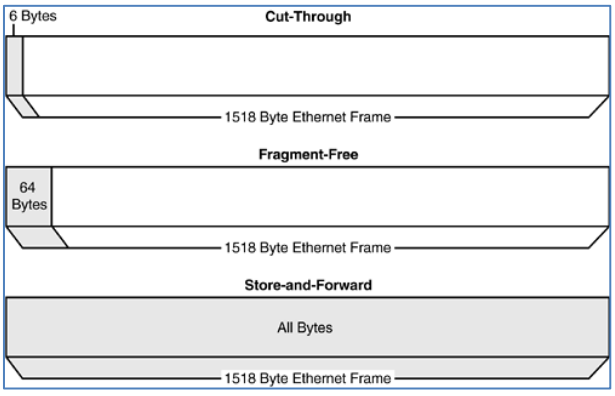

Ilustración 3 - Tipo de procesado de entrada en los conmutadores

Independientemente de la cantidad de *bytes* de cada trama que sean examinados, se debe de conmutar los *frames* desde el puerto de entrada a uno o más puertos de salida. Para ello se utilizará la denominada la matriz de conmutación (*switch fabric*) con la que se establecerá comunicaciones entre los diferentes canales.

Hay varias formas de implementar matrices de conmutación en los switches. Entre ellas se destacan dos grandes grupos: la conmutación por división de tiempo (*time-division switching*) y la conmutación por división de espacio (*space-division switching*).

En el primer grupo (división por tiempo) las celdas de los diferentes puertos de entrada están multiplexadas y transmiten datos a partir de un único canal. Esta estructura estará limitada por el ancho de banda de comunicación interna de dicho canal; este mecanismo presenta la ventaja de que no se ve afectado por problemas de bloqueo.

Por otro lado, los conmutadores de división por espacio tienen una estructura en la cual múltiples rutas de datos están disponibles entre los puertos de entrada y los de salida. Por lo que los datos de celdas de diferentes puertos de entrada y de salida pueden ser retransmitidos concurrentemente. La capacidad de conmutación total será por lo tanto el producto del ancho de banda de cada ruta de acceso con el número de caminos que permiten cursar datos de forma simultánea. Esta clase de matriz de conmutación se subdivide según el número de rutas disponibles entre cada par de puertos de entrada/salida. Los *single-path switches* serán aquellos en los que existe un único camino entre los pares de puertos; mientras que los *multiple-path switches* serán aquellos en los que existirá más de una. El primer tipo tendrá un control de enrutamiento más sencillo de utilizar, mientras que los segundos serán más complejos, pero más flexibles y más tolerantes a fallos de bloqueo y de contención.

Conmutación por división en el tiempo	
Shared-memory	Las celdas que llegan a los puertos de entrada son multiplexadas en un medio de alta velocidad como un bus o un anillo. Se utilizará un filtro de direcciones y un *buffer* FIFO para cada línea de salida del medio compartido.
Shared-medium	Es similar al anterior, con la salvedad de que los puertos de entrada serán multiplexados en un *buffer* de memoria compartida que se ha de implementar con una memoria de alta velocidad. Tendrán un planificador encargado de demultiplexar los datos en el puerto de salida pertinente.
Conmutación por división en espacio	
conmutadores de camino único (*single-path*)	
Crossbar	Consistirá en un *array* de dos dimensiones en los que se comunicará cada par de puertos de salida y entrada.
Fully interconnected	Se basa en una comunicación *full mesh* entre los puertos de entrada/salida, es decir cada puerto de entrada tendrá una ruta exclusiva para llevar datos a cada uno de los puertos de salida.
Banyan	Son una familia de conmutadores que construyen sus rutas a través de elementos básicos de conectividad 2 x 2 entre puertos, existiendo tan solo una única ruta entre cada par puertos de salida y entrada.
Multiple-path switches	
Aumented banyan	Es un conmutador de *Banyan* con más etapas, las cuales aumentan la probabilidad de que la celda sea enrutada correctamente a su destino.
Closs	Consiste en una conmutación en tres fases: en la primera se distribuye el tráfico; en la segunda se construyen varios módulos de conmutación para proporcionar varias posibles rutas a seguir y en la última etapa las celdas son transmitidas a su puerto de salida correcto.
Multiplane	Tienen múltiples planos independientes de rutas entre los puertos de entrada y de salida en el conmutador.
Recirculation	Se reintroducen en los puertos de entrada las celdas a transmitir que no han sido transmitidas en los puertos de salida.

Tabla 2 - Tipos de matrices de conmutación (*switch fabric*)

Por lo tanto, si se trabaja con conmutadores por subdivision en el tiempo, las tramas deberán esperar a ser transmitidas hasta que llegue su turno.

En el caso de conmutadores por división en espacio se producirán retrasos cuando exista congestión en la matriz de conmutación.

Se necesita entonces un mecanismo de almacenamiento para que estos paquetes esperen y no sean descartados tanto en la salida como en la entrada del dispositivo. Estas serán las colas de entrada y salida que serán implementadas en los *bufferes* de almacenamiento del conmutador. Existen difrentes estrategias para su implementación:

- ***Colas de memoria compartida:*** Se utiliza un único *buffer* para implementar las diferentes colas de entrada y de salida para cada puerto. Es típico en conmutadores por división de tiempo.
- ***Colas de salida.*** Se implementan una serie de colas de salida tras la matriz de conmutación donde se almacenará los paquetes a ser enviados. En ellas se pueden implementar políticas de calidad de servicio.
- ***Colas de entrada.*** Se implementan un serie de colas de entrada donde se almacen los frames antes de que entren en la matriz de conmutación. Tienen asociado el problema del bloqueo HOL (*head-of-the-line*), que se producirá cuando el primer elemento de la cola de entrada no pueda ser conmutado (por estar ocupada la cola del puerto de destino); este bloqueo afecta a los siguientes paquetes que podrían ser cursados ya que su puerto de destino podría estar libre.
- ***Colas VOQ (virtual output queues).*** Implementación de multiples colas virtuales en los puertos de entrada (también se pueden implementar en las colas de salida) para eliminar el problema de HOL. Con esto tenemos que, la memoria de entrada correspondiente a cada puerto se subdivide en tantas colas virtuales como puertos de salida, produciendo que un paquete cuyo destino está bloqueado no afecte a la transmisión de los siguientes paquetes que accedan al conmutador y cuyo puerto de destino esté libre.
- ***Crosspoint queues***. Se trata de establecer colas en búferes de almacenamiento situados en los cruces de una malla

Los búferes y colas implementados en los conmutadores serán ocupados en mayor o menor medida según la existencia de:

- Desajustes en la velocidad de entrada y salida
- Competencia de tráficos distintos por un mismo puerto de salida
- En casos de colisiones *half-duplex* en el puerto de salida.

Uno de los principales problemas de los conmutadores son los bloqueos en la ruta de comunicación en la matriz de conmutación. Esto ocurrirá solo en los conmutadores por división en espacio, debido a que los que están divididos por tiempo multiplexan el tráfico con el objetivo de eliminar este problema. Que un conmutador sea libre de bloqueo significará que siempre habrá camino entre un puerto de salida y otro de entrada cuando éstos estén ociosos.

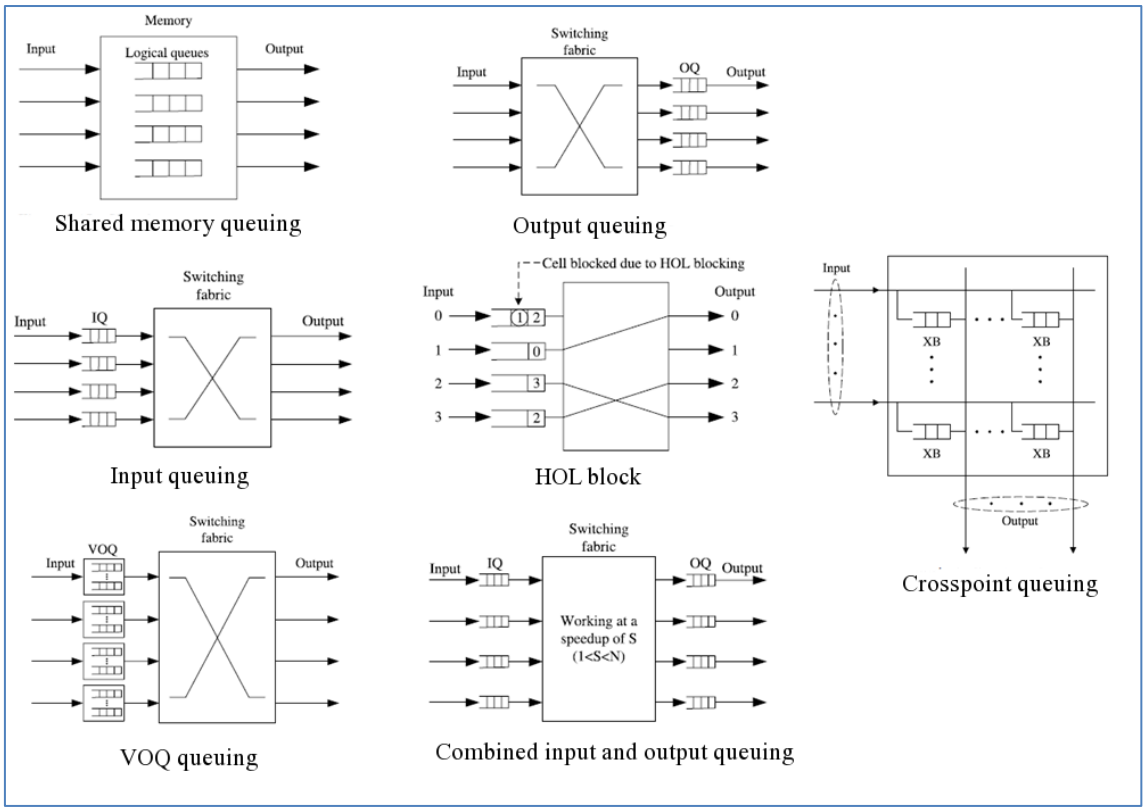

Ilustración 4 - Implementaciones de colas en los búferes de los conmutadores

Otro problema es la contención de la salida que ocurrirá principalmente cuando dos o más puertos de entrada quieran acceder a la misma salida. Ocurrirá principalmente en los conmutadores por división de espacio, ocurriendo en los temporales solo cuando el número de paquetes enviados secuencialmente a la salida supera a la capacidad del buffer en un periodo de tiempo determinado.

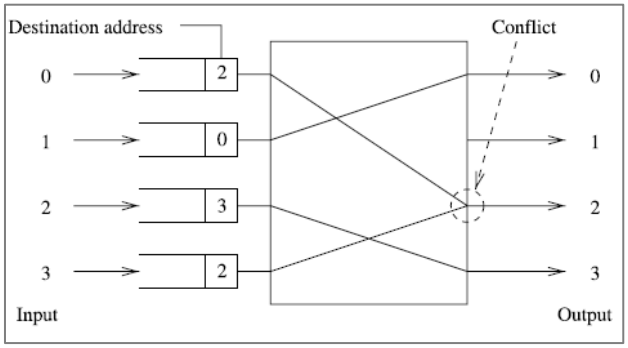

Ilustración 5 - Ejemplo típico de contención de salida

Por último, una vez que las tramas se almacenan en las colas de salida sólo tenemos que entregarlos al puerto para enviarlo a su destino. Como indicamos anteriormente existen varias colas para cada salida; los conmutadores implementan diferentes tipos de planificadores para establecer qué cola y qué paquete se transmite en cada momento. En el *"Capítulo 4 – Calidad de servicio"* se explicará con algo más de detalle los diferentes tipos de planificadores.

4.1.1 Arquitectura específica de los conmutadores Juniper EX4200 y comparación con el EX8200

El conmutador EX 4200 del fabricante Juniper es un dispositivo de conmutación de tráfico de propósito general. Dispone de 24 puertos de 1 gigabit ethernet, junto con 2 puertos de 10 gigabit ethernet y que analiza las tramas *ethernet* en modo *store-and-forward mode* (analiza en la entrada, todos los bytes del frame).

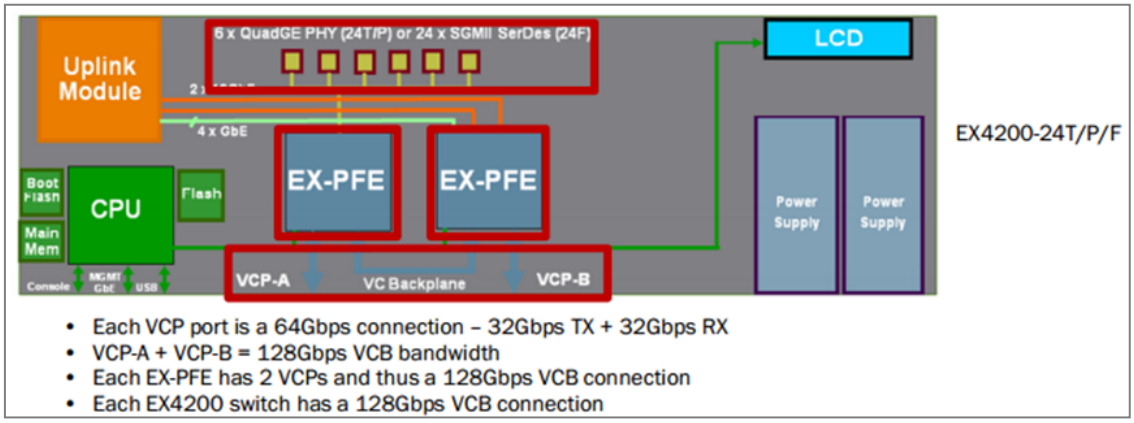

Ilustración 6 - Arquitectura EX 4200

Lo más relevante de su arquitectura es que, como se aprecia en la imagen, sus puertos de entrada y salida están conectados a dos componentes denominados *EX – PFE* (*packet forward engine*), uno encargado de gestionar los 24 puertos de 1 *gigabit*, y el otro encargado de gestionar los de 10 *gigabit*. Este es un componente clave en el dispositivo, debido a que cada componente PFE funciona como un pequeño conmutador de memoria compartida (cuyo tamaño será de 3MB) en el que se implementa una *buffer* compartido, que junto a la lógica de este chip ASIC, funcionará como una matriz de conmutación en la que se implementarán tanto búferes para puertos de entrada, como para puertos de salida.

La vida de una trama conmutada por este dispositivo es la siguiente: en primer lugar éste llega a un puerto de entrada, a partir del cual es transportado por el *midplane* hasta el dispositivo *PFE*. Una vez allí, este es almacenado en el buffer del puerto de entrada de la memoria compartida, en el cual la lógica de control del componente irá atendiendo a las diferentes tramas mediante una planificación FIFO. Por lo tanto las tramas irán siendo procesadas secuencialmente por la lógica del dispositivo (de ahí que se diga que es como un conmutador por división de tiempo). En el procesamiento de entrada se le extraerá la cabecera, se analizará su campo VLAN, se le aplicarán políticas de *firewall*, *policy*; y en definitiva se extraerá la MAC. Una vez se sabe el destino al que tiene que ir, se introduce en una determinada cola de salida, sufriendo los típicos mecanismos de QoS de encolado, planificación, control de tráfico, etc; hasta que finalmente es enviado a su destino.

Los PFE están libres de bloqueos, debido a que las tramas son servidas desde las colas de entrada a las de salida secuencialmente. Y solo existirá contención cuando se envíen más tramas de las que caben en la memoria en un determinado período de tiempo.

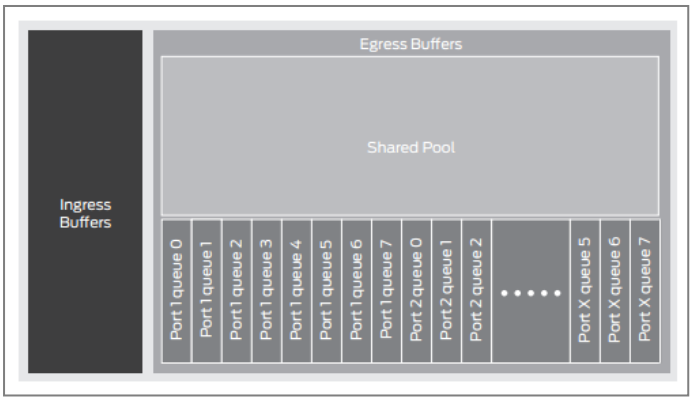

Ilustración 7 - Memoria compartida existente en cada PFE del Juniper EX 4200

En contraposición el equipo de gama superior EX8200 cuenta con búferes de entrada dedicados, matrices de conmutación sin bloqueo (*crossbar*) y búferes de salida de gran tamaño; por el contrario los equipos considerados de acceso cuentan con memorias de entrada compartidas para todos los puertos y ASICs de propósito específico que son los que propiamente realizan la conmutación de paquetes. En éstos últimos los búferes de salida son de menor tamaño, por lo que dividen dicha memoria de salida en dos partes; una de ella compartida por todos los puertos y otra para crear los búferes de salida dedicados por puerto[2].

En el caso de los EX4200, como se puede ver en la figura, existirán ocho colas virtuales por cada puerto físico de salida. El componente *shared-pool* viene a representar una memoria compartida por todas las colas, que podrá ser configurada para que el tráfico más prioritario tire de ella cuando exista congestión en la cola en la que se desea almacenar. Es un mecanismo con cierta similitud al visto en la literatura en el documento "***Broadcom Smart-Buffer Technology in Data Center Switches for Cost-Effective Performance Scaling of Cloud Applications***" ([08] Das & Sankar, 2012)

¿Se puede decir entonces que Juniper EX 4200 es un conmutador de memoria compartida? No exactamente, principalmente debido a la propiedad virtual chasis. En el anterior esquema de la arquitectura se puede observar que existen unos buses que salen de los PFE para comunicarlos entre sí, y otros que salen del conmutador (llamados en el dibujo VCP-A). Estos son las conexiones que proporcionan el llamado chasis virtual, a partir del cual mediante las diferentes combinaciones con otros elementos de la gama Juniper EX, se puede crear una matriz de conmutación compleja (que esta vez sí será por división de espacio y necesitará prestar atención a los bloqueos).

Por último mencionar las políticas de calidad de servicio que soporta este conmutador: planificación de las colas SP y SDWRR, gestión de la contención mediante el algoritmo WTD y clasificación del tráfico tanto por el protocolo IEEE 802.1p como por el DSCP.

4.2 Aplicaciones y protocolos de comunicaciones

Como se ha dicho anteriormente, en la actualidad la ejecución de trabajos para HPC se hace en una infraestructura dedicada y específica para este cometido utilizando redes de interconexión de alta velocidad y coste con latencias de datos muy bajas. Algunas de estas tecnologías especializadas son *Infiniband* o *Myrinet*. Por el contrario en los *clouds* genéricos

[2] Véase los conceptos de QoS relevantes en la sección "QoS en conmutadores" en pág 22

conviven con tráficos de diferente naturaleza se utiliza una tecnología de red más económica y más difundida como es *1 gigabit ethernet*

Utilizar la tecnología *ethernet* y TCP/IP para aplicaciones de HPC como las implementadas por MPI (conocida librería de paso de mensajes utilizada normalmente en entornos distribuidos para computación de altas prestaciones) sufrirá problemas de rendimiento debido a múltiples razones.

Por ejemplo, en el caso de una aplicación MPI que utiliza de manera distribuida dos servidores conectados mediante un conmutador los mensajes que intercambiarían emisor y receptor serían enviados desde la capa de aplicación, por lo tendrían que ser troceados en paquetes para ser enviados por el protocolo TCP/IP a través de la capa de transporte y de la de red. A su vez, estos paquetes serían divididos en tramas *ethernet* para ser enviados a través de la capa de enlace; una vez éstos llegan al receptor éste tendrá que realizar el proceso inverso hasta llegar a la capa de aplicaciones para que el receptor MPI procese el mensaje según su necesidad.

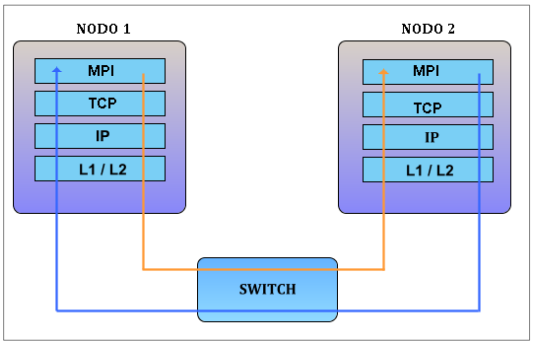

Ilustración 8 - Ejemplo de comunicación en la pila TCP/IP para programas MPI

Para solucionar el problema de la pila TCP o más genéricamente del excesivo número de capas del modelo OSI, típicamente se utilizan las técnicas del *kernel-bypass* y *zero-copy*. Es de relevancia comentar que las implementaciones MPI de los protocolos de comunicaciones de *Infiniband* y *Myrinet* utilizan estas técnicas para que las comunicaciones pierdan el menor tiempo posible en el kernel del sistema operativo.

- ***kernel-bypass:*** mediante esta técnica, las aplicaciones del espacio de usuario se comunican directamente con el subsistema de entrada/salida que gestiona la comunicación de red (en este caso la capa de enlace). Es decir, se salta la capa de transporte y de red que normalmente viene a estar representada por protocolos como TCP/IP. Esto disminuye mucho los retardos de red al tener que traducir a menos capas los datos a transmitir.
- ***zero-copy:*** son operaciones que requieren el apoyo de hardware y software específico, en las cuales la CPU no tiene que realizar la tarea de copiar datos de una posición a otra de la memoria, reduciendo los cambios de contextos y aumentando la eficiencia. Son especialmente importantes en las redes de alta velocidad donde la capacidad de un enlace de red se acerca o excede a la capacidad de procesamiento de la CPU (para evitar que la CPU gaste casi todo su tiempo copiando los datos transferidos).

Las conexiones TCP (en la típica implementación RENO) se componen de tres etapas: *establecimiento de la conexión*, *transferencia de datos* y *fin de la conexión*. En la primera se realiza una negociación en tres pasos para establecer la conexión propiamente dicha: el receptor abre un puerto enviando un paquete SYN, el receptor valida la conexión devolviendo un paquete SYN/ACK, con lo que el emisor finalmente le mandaría un ACK al receptor completando el establecimiento de la misma.

En la *transferencia de datos* se enviarán paquetes de forma fiable, recibiendo el emisor un ACK (*acknowledgement*) para que tenga constancia de que este ha sido recibido por el receptor. Si un ACK de un dato no es devuelto (vencimiento del temporizador asociado), el emisor puede interpretar que hay congestión en la red y puede disminuir drásticamente su tasa de envío de datos. Esto será gestionado por los mecanismos de control de la congestión y sus algoritmos. Una mejora de TCP Reno (como por ejemplo TCP New Reno), llamada asentimiento selectivo (SACK, *selective acknowledgement*) permite a un receptor asentir los datos que se han recibido de tal forma que el remitente solo retransmita los segmentos de datos que faltan.

En TCP el *throughput* de la comunicación está limitado por dos tamaños de "ventana": la ventana de congestión y la ventana del receptor. La primera trata de limitar la transmisión para no exceder la capacidad de la red (control de congestión) y la última tiene el mismo propósito en relación a la capacidad del receptor para procesar datos (control de flujo). Cada segmento TCP contiene el valor actual de la ventana de recepción. Si el emisor, por ejemplo enviase un *ACK* que confirma el byte 4000 y especifica una ventana de receptor de 10000 (bytes), el emisor no transmitirá ningún paquete adicional al 1400, incluso aunque la ventana de congestión lo permita.

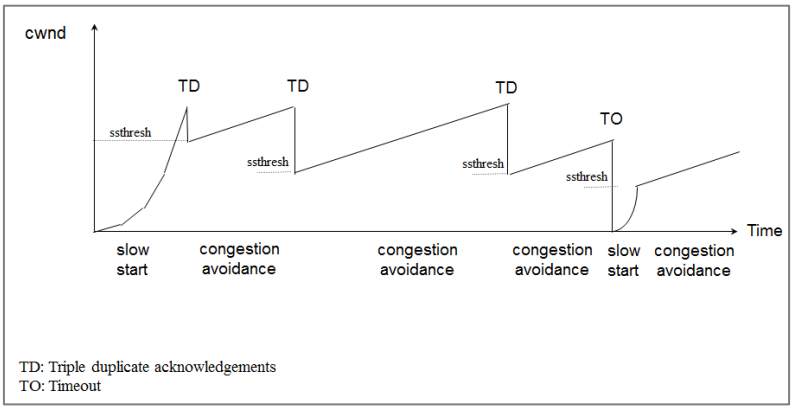

Ilustración 9 - Variaciones de tamaño de la ventana de congestión en TCP

La implementación de TCP RENO es una de las primeras variantes y sentó las bases de los mecanismos de las actuales pilas TCP descritos a continuación:

- ***Arranque lento (slow start):*** Debido a que no se tiene la capacidad de saber el volumen máximo de datos que se puede transmitir en la red; este algoritmo comenzará enviando un volumen de datos pequeños que irá aumentando hasta que la red sature.
- ***Control de la congestión (congestion control):*** Se detectará que la red está saturada cuando venza un temporizador de retransmisión de un paquete enviado. Este mecanismo será el encargado de poner el valor de la ventana de congestión (*cwnd*) a MSS (*maximum segment size*), con lo que se volverá a utilizar de nuevo el algoritmo del arranque lento.
- ***Retransmisión rápida (fast retransmit):*** Este es una mejora en el mecanismo de congestión de TCP para diferenciar el trato en otro tipo de congestión: el receptor detecta que le falta un paquete, por lo que le envía al emisor un ACK duplicado; si esto vuelve a ocurrir le enviará de nuevo otro ACK. Cuando el emisor recibe un ACK triplicado vuelve a enviar el paquete aún sin esperar a que venza su temporizador. Se considera un problema de congestión leve, por lo que la reducción en la ventana de congestión será mucho menor que en el anterior caso. A continuación la ventana de congestión se incrementa linealmente, con cantidades pequeñas, probando cautelosamente el ancho de banda disponible.

Esta forma de actuar ante la congestión tiene un especial impacto cuando compiten diferentes tráficos por los mismos recursos. El protocolo TCP siempre va a tratar de llenar la cola del puerto de salida en el conmutador, con el objetivo de obtener el mayor ancho de banda posible. Tarde o temprano uno o más tráficos de este tipo producirán congestión de los búferes de los conmutadores provocando descarte de paquetes en éstos y el vencimiento de temporizadores en los emisores. Esto producirá incrementos importantes de latencia de todos los flujos de tráfico que tengan que pasar por misma cola del puerto congestionado. En el escenario de un cloud privado el RTT (*round trip time*) entre pares de máquinas es habitualmente muy bajo, por lo que se el protocolo TCP incrementará rápidamente la ventana de transmisión produciendo a su vez congestión de los búferes del conmutador. También habrá que tener en cuenta que se convivirán con protocolos independientes del RTT, como UDP los cuales no tienen mecanismos de control de congestión, y también existe peligro de que llenen los búferes de entrada y salida de las NIC; o bien las colas implementadas en el conmutador.

Adicionalmente a la de RENO existen otras variantes de los algoritmos de control de congestión que implementan mejoras en los diferentes algoritmos aliviando parte de las problemáticas identificadas para este protocolo (ineficiencia en entornos con pérdida de paquetes bajas por ej.). A continuación se enumeran algunas de las más conocidas:

- ***Vegas**:* es un algoritmo de control de congestión que aumentará/disminuirá la ventana de congestión a partir del retardo que experimentan los paquetes (es decir se basa en el RTT). Por lo que detectan congestión antes de que se pierdan paquetes.
- ***Bic:*** algoritmo de congestión optimizado para redes de alta velocidad con altas latencias. Este intentará obtener el máximo tamaño de ventana de congestión en la cual permanezca estable durante más tiempo, mediante un algoritmo de búsqueda binario.
- ***Cubic:*** es un derivado menos agresivo que BIC en el que el incremento de la ventana vendrá determinado por una función cúbica del tiempo transcurrido desde el último evento de congestión. Es decir, no se basa en la recepción de ACK para aumentar el tamaño de la ventana. En contextos de RTT muy cortos, recibirá ACKs muy rápido, por lo que las ventanas de congestión crecerán muy rápido. CUBIC ofrece la ventaja de que su crecimiento es independiente del RTT.
- ***HSTCP (high speed TCP):*** es un algoritmo que modifica el control de congestión del estándar TCP para que se utilice plenamente el ancho de banda en redes da alta velocidad y grandes latencias. Esto significa que la ventana de congestión crecerá más rápido que en el estándar de TCP, y también se recuperará de las pérdidas más rápidamente.
- ***DCTCP:*** Es también una mejora del algoritmo de control de congestión TCP para redes de centros de datos. Utiliza el protocolo ECN (*explicit congestion notification*) el cual es una extensión de los protocolos IP y TCP que permiten notificar de extremo a extremo la presencia de congestión en la red, sin tener que llegar a descartar paquetes. De este modo el protocolo reacciona a la congestión a medida que se va presentando, y no cuando ya ha se ha producido, como ocurre en las versiones clásicas de TCP.

Por último, TCP *finaliza la conexión* en una negociación por cuatro pasos, mediante el envío de un segmento FIN y su correspondiente ACK por cada uno de los puntos de la comunicación.

4.3 Tecnologías de virtualización

El despliegue de un *cloud* privado conlleva el uso de técnicas como la virtualización de servidores, siendo dichos servidores los emisores/receptores del tráfico en los escenarios ya mencionados y resultando punto crítico a analizar para el estudio de la convivencia de diferentes tipos de tráfico en la infraestructura local. En el presente apartado explicaremos los conceptos básicos de virtualización así como los mecanismos necesarios para la transmisión de información en el caso concreto de Xen Hypervisor.

En los sistemas virtualizados existirán dos tipos diferentes de máquinas: por un lado tendremos los *"host"* (también llamados **dom0**), que representará al anfitrión que se encargará de controlar las diferentes instancias de MV creadas; y por otro lado estarán los *"guest*" (también llamados **domU**), que representarán a los huéspedes. Existen múltiples formas de implementar virtualización en los servidores, entre las que se destacan:

- **Para-virtualización:** En esta técnica se modifica el núcleo del sistema operativo de la máquina virtual huésped para que se pueda ejecutar específicamente sobre el hipervisor, el cual es un programa que se encargará de ejecutar directamente sobre el hardware las diferentes instrucciones de los *"guest"* sobre el anillo 0 del sistema operativo anfitrión. El rendimiento es muy alto debido a que el hipervisor utiliza las instrucciones de la máquina física directamente (no existe traducción binaria de instrucciones).
- **Virtualización Completa:** En la cual no se necesita modificar nada del sistema operativo de las máquinas virtuales huéspedes, ni en el anfitrión. En este caso el hipervisor emulará la CPU física para manejar y modificar las diferentes operaciones que necesitan realizar los sistemas operativos huéspedes. Debido a esta emulación, el rendimiento en este caso será peor, debido al proceso de traducción de instrucciones que tendrán que sufrir las diferentes acciones de las máquinas virtuales "*guest*".

Tradicionalmente los sistemas paravirtualizados proporcionaban mejor rendimiento en procesamiento de tareas y acceso a los dispositivos, sin embargo esta afirmación ha ido cambiando a lo largo de los últimos años mediante mejoras como pueden ser la utilización de drivers paravirtualizados y soporte a la virtualización por hardware (HVM).

La comunicación entre las diferentes máquinas virtuales alojadas en un mismo servidor es un punto crítico objeto de análisis.

Existen varias formas de configurar las tarjetas de red de las máquinas virtuales. En primer lugar está el modo **NAT**, en el cual se crea una red virtual totalmente aislada dentro del servidor físico. Para establecer una comunicación entre las máquinas de esta red con otras que estén en el exterior, se hará a través de una especie de firewall dentro de la aplicación de virtualización. Por lo tanto, existirá un proceso de traducción entre las IP privadas de la red virtual, con las del exterior.

En segundo lugar está el modo ***bridge***, con el cual se puede extender la red a la que está conectada el servidor físico a las tarjetas virtuales de cada *domU* (por lo que cada una de ellas será un *host* más en la red a extender). Por lo tanto, para el servidor huésped *dom0* existiría un bridge para cada máquina virtual que esté desplegada en él. Como en la virtualización de servidores, debido a la existencia de tráfico para HPC, el modo bridge será el más adecuado a utilizar en el presente estudio. Principalmente debido a que no se produce una traducción de direcciones IP como en NAT, que sin duda nos daría problemas de latencia en las comunicaciones.

Las diferentes máquinas *domU* desplegadas en un sistema no tendrán acceso directo a los dispositivos hardware (incluyendo aquí a la tarjeta de red) de la máquina física. El único que tendrá un sistema operativo privilegiado para acceder a estos recursos físicos será el *dom0*.

Por lo tanto, el domU tendrá que comunicarse con el dom0 para acceder a los dispositivos. Esta comunicación se hará a través de las interfaces *front-end* como el *netfront* y *blockfront* (para operaciones de red y disco respectivamente). Estas interactuarán con sus respectivos drivers situados en el dom0 que controlan los dispositivos. La transferencia de datos se hará a través de una memoria compartida con sus homologas interfaces *back-end*. Los eventos relacionados con la transacción se comunican a través por un canal de eventos del dominio, el cual conecta a cada interfaz *front-end* con su correspondiente interfaz back-end, la cual a su vez gestiona el acceso a correspondientes drivers que forman parte del sistema operativo anfitrión, es decir el dom0.

Estos driver de *back-end* son los responsables de la planificación de operaciones de E/S de los diferentes *domUs*.

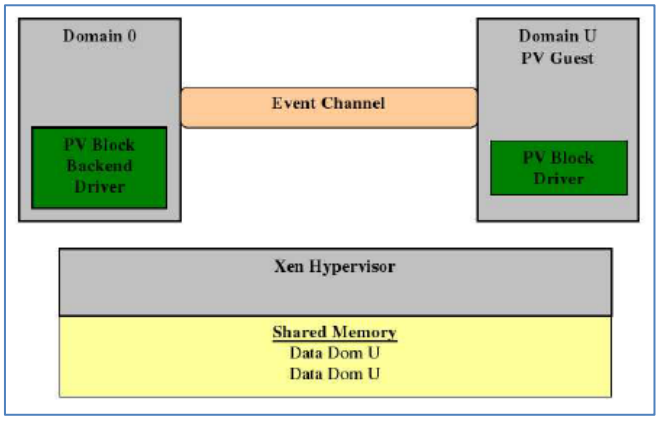

Ilustración 10 - Comunicación entre los domU y los dom0 para el acceso a los dispositivos hardware

El control de la transferencia de información de Xen consiste en un conjunto de *eventos ligeros* e *hiperllamadas asíncronas*, que emularan las interrupciones en los dispositivos y llamadas al sistema en un sistema operativo normal.

La comunicación de Xen con un dominio se realizará a través de un canal de eventos asíncrono, que se utilizará para reenviar interrupciones del dispositivo a los dominios a través del canal de eventos. Puesto que este mecanismo es asíncrono, las llamadas no son atendidas de inmediato, por lo que se almacenarán con una máscara de bits para identificar de qué domU pertenecen a la espera de ser planificadas.

En el proceso de comunicación inverso, entre un domU y Xen, este último exportará una interfaz de hiperllamadas para dominios con el objetivo de hacer privilegiadas las operaciones con los dispositivos. Estas *hiperllamadas síncronas* son análogas a las llamadas al sistema en sistema operativo no virtualizado.

La comunicación entre las interfaces de *front-end* y *back-end* ocurrirá a través de una capa adicional llamada Xen i/O ring. Este anillo es una cola circular que contiene los descriptores de E/S que referencian a los datos que hay que transferir; lo cual seguirá un esquema productor-consumidor en el cual existirán cuatro actores:

- *El productor peticionario*: Coloca las solicitudes de descriptores de E/S en el anillo.
- *El consumidor peticionario:* Consume las solicitudes de descriptores de E/S en el anillo.
- *El productor de respuestas:* Coloca los mensajes de respuesta de los descriptores de E/S en el anillo de la correspondiente petición.

- *El consumidor de respuestas:* Consume los mensajes de respuesta de los descriptores de E/S situados por el productor de respuestas.

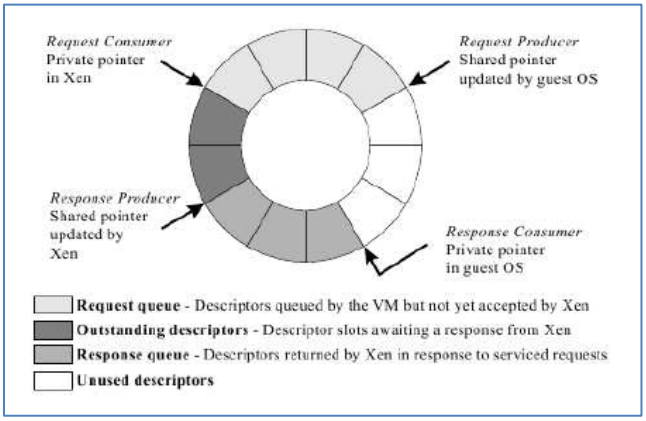

Ilustración 11 - Anillo de memoria compartida utilizada para la comunicación entre los domU y los dom0.

Este mecanismo es asíncrono, eficiente y lo suficientemente genérico como para soportar la gestión de acceso de diferentes dispositivos (incluidos los de red). Por lo tanto, los datos son escritos en la memoria compartida entre los *domU* y los *dom0*, los cuales estarán referenciados por sus respectivos descriptores de E/S. Los *domU* asocian un único identificador, el cual está situado en los descriptores, y que es reproducido por su respuesta asociada. El acceso a cada anillo será entonces realizado por dos pares de punteros productores y consumidores. Los dominios sitúan sus peticiones en el anillo de E/S mediante la emisión de una hiperllamada a XEN, y una vez estas peticiones hayan sido atendidas por el planificador del dispositivo, serán recogidas por los dominios de destino, con lo que tendrá la posición de memoria específica donde recoger el dato en la memoria compartida.

4.4 Linux y los dispositivos de red

Por último se explicará el funcionamiento de Linux y los dispositivos de red *ethernet* en los sistemas anfitriones (máquinas físicas).

Los interfaces de red normalmente disponen de una serie de búferes que se utilizan para almacenar las cabeceras IP y los datos según se describe en el artículo *"A 10 Gigabit Programmable Network Interface Card* ([31] Willmann, 2008). Adicionalmente poseen también una cola donde se almacena los descriptores de búferes, en los que se indica posición de memoria del buffer y el tamaño de éste. Como se verá en los siguientes pasos, esta será la unidad de transacción entre el sistema operativo y la NIC. Para el envío de una trama *ethernet* se llevarán a cabo los siguientes pasos:

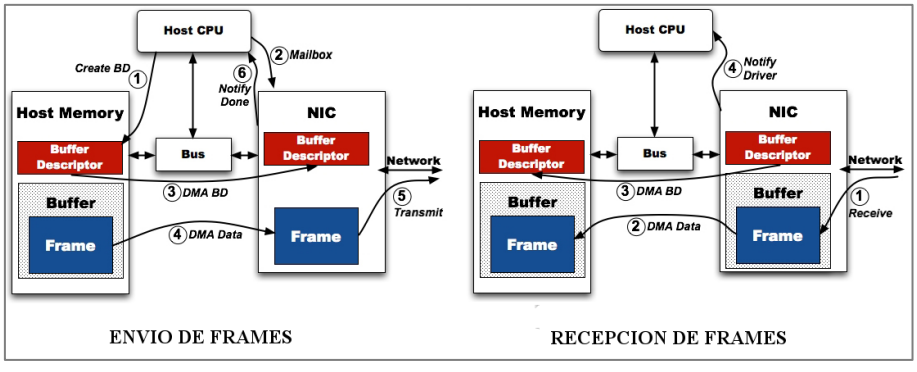

Ilustración 12 - Envío y recepción de paquetes en una NIC Gigabit Ethernet

1. El sistema operativo anfitrión es informado de que una trama que está almacenada en memoria del host está lista para ser enviada. Se construye el descriptor de *buffer* de la trama en cuestión.
2. El sistema operativo notifica a la NIC de que un nuevo descriptor de *buffer* está en la memoria del *host* y está listo para ser procesado. A este evento se llama normalmente como un evento "*mailbox*".
3. La NIC inicializa la DMA (*direct memory access*) y se lee el descriptor del *buffer* de memoria.
4. A partir de la memoria del descriptor de *buffer* se lee ahora la trama a la tarjeta de red.
5. Cuando todos los segmentos de la trama han llegado, la NIC la transmite a la capa de red para ser transportada.
6. Dependiendo de cómo esté configurada la NIC, esta puede interrumpir al sistema operativo para indicarle que el envío del frame se ha completado.

Por el contrario, para la recepción de una trama se seguirán los pasos indicados a continuación, presuponiendo que el sistema operativo ha creado los descriptores de *buffer* que apuntan a regiones de memoria que están libres, donde se almacenarán los nuevos datos a recibir:

1. La NIC recibe la trama (la almacena en su *buffer*) que le es enviada desde la capa de red.
2. La NIC inicia una escritura DMA del contenido de la trama en la memoria libre del *host* (lo mete en donde indica el descriptor de *buffer* del host que apuntaba hacia memoria libre).
3. La NIC modifica el descriptor de fichero de esta nueva trama para introducir el espacio que ocupa. A continuación copiará este descriptor de fichero a la cola de descriptores de fichero del *host* con un proceso DMA.
4. Igual que en el caso anterior, dependiendo como esté configurada la NIC, esta interrumpe a la CPU del host para indicarle que la llegada de la trama se ha completado.

Por último es importante mencionar los mecanismos que implementan las tarjetas de red a día de hoy para la mejora de las transferencias y su posible influencia en el escenario analizado:

- ***Checksumming:*** El *kernel* de Linux da la opción de no calcular el *checksum* de los protocolos TCP y UDP.
- ***Scatter-gather I/O:*** Método por el cual en vez de estar trabajando con gran número de búferes grandes llenos de datos, se transmitirán pequeños búferes que formen parte de un *buffer* lógico mayor.
- ***Large send offloading/TCP segmentation offload:*** Es una técnica para aumentar el rendimiento de salida de las conexiones de red mediante la reducción de carga de la CPU. Grandes cantidades de datos encoladas en los búferes son subdivididos por la NIC en segmentos más pequeños con el objetivo de pasar a través de todos los elementos de red.
- ***Large receive offload:*** Se trata también de una técnica para aumentar el rendimiento de salida de las conexiones de red mediante la reducción de carga de la CPU. Este mecanismo funciona agregando múltiples paquetes entrantes desde un único flujo que tiene asignado un tamaño de buffer muy grande, reduciendo así el número de paquetes que se tienen que procesar. TCP en Linux solo acepta su implementación software.

- ***Interrupt coalescing:*** es una técnica que está implementada en las NIC por la cual se permite la recepción de un grupo de tramas, teniendo que notificar al *kernel* del sistema operativo mediante una única interrupción. Esto también aumenta el rendimiento de salida de las conexiones de red mediante la reducción de carga de la CPU.

También existen otros parámetros del *kernel* de Linux que deben tenerse en cuenta:

- ***Tcp low latency*** (tcp_low_latency): Cuando este parámetro está habilitado la pila TCP toma decisiones que priman la latencia baja sobre el throughput elevado. Por defecto esta opción está deshabilitada primando el comportamiento orientado a este último. Lo que este parámetro hace en realidad es deshabilitar el procesado de la pre-cola TCP. TCP tiene tres colas TCP: la cola de recepción, la cola de backlog y la precola. Deshabilitando tcp_low_latency provocará que se realice un bypass de la precola y éstos vayan directamente a la cola de recepción.
- **Búferes de los protocolos y de las aplicaciones**: Los tamaños de los búferes por defecto envío y recepción de TCP/UDP así como los tamaños de las colas de transmisión y recepción (*txqueuelen* y *backlog*).

4.5 Calidad de servicio

El término *calidad de* servicio se refiere a la capacidad de la red de proporcionar mejor servicio a un tráfico de red seleccionado sobre otros. El objetivo principal de éste es proporcionar priorización incluyendo ancho de banda dedicado, *jitter* y latencia controlados, así como características de pérdida de paquetes mejoradas.

En los siguientes apartados se indicarán tanto las métricas a medir, como una serie de técnicas aplicadas en servidores y conmutadores que tienen como objetivo ofrecer soluciones a problemas de congestión. Al aplicar QoS es importante ser consistente: se ha de tener en cuenta que las políticas escogidas han de ser coherentes con todos los elementos de la infraestructura, puesto que, de no ser así, el resultado de estas no será apreciable en la calidad global del servicio ofrecido.

4.5.1 Métricas de QoS

Se exponen a continuación las métricas que serán relevantes en el escenario que nos ocupa:

- ***Ancho de banda:*** El ancho de banda es la cantidad de información o de datos que se puede enviar a través de una conexión de red en un período determinado. Normalmente este se expresa en bits por segundo (bps) o por sus múltiplos: kilobits por segundo (Kbps), megabits por segundo (Mbps) o gigabits por segundo (Gbps). Este término también puede referirse tanto a la *capacidad de ancho de banda*, lo cual significaría la máxima salida de datos en un sistema de comunicación digital; como al *ancho de banda consumido*, lo cual se corresponde al rango promedio de datos descargados a través de la red.
- ***Retardo unidireccional:*** Es el retardo existente desde un extremo a otro en un solo sentido de la comunicación. Es decir es el tiempo que tarda en llegar un paquete a su destino. Los retardos unidireccionales desde un nodo A un nodo B pueden ser medidos enviando paquetes con impresión horaria (*timestamped*) desde A, y registrando los tiempos de la recepción en B. La dificultad es que A y B necesitan tener sus relojes sincronizados. Esto se

puede conseguir usando técnicas como el sistema de navegación universal (GPS) o usando también el *Network Time Protocol* (NTP). Esto principalmente se hace debido a que en comunicaciones a grandes distancias los paquetes no tienen por qué ir y volver por la misma ruta.

- ***Jitter:*** Este parámetro también llamado variación del retardo es una métrica que describe el nivel de desviación del tiempo de llegada de los paquetes con respecto a un patrón ideal. Normalmente dicho patrón será el mismo a través del cual fueron enviados los paquetes. Tales disturbios pueden ser causados por tráfico de competencia, o por contención en el procesado de recursos en la red. El *RFC 3393* define una métrica de variación de IP, en la cual se compara los retardos experimentados por paquetes de igual tamaño, y considerando que el retardo es naturalmente dependiente del tamaño del paquete. A diferencia del retardo unidireccional, la variación de retardo se puede medir sin necesidad de sincronizar los tiempos de los nodos a analizar. Normalmente las herramientas que miden el retardo unidireccional, también miden la variación de retardo o *jitter*.

También existen otros parámetros de red interesantes en el ámbito de las comunicaciones como son la pérdida de paquetes o particulares del protocolo de transporte TCP como es el reordenamiento de paquetes o el factor de duplicación de estos.

4.5.2 QoS en servidores

Para el caso en estudio ocurrirá que convivirán en la misma infraestructura de red y sistemas flujos sensibles a latencia (HPC) con flujos de servicio sin estas restricciones. En el presente apartado analizaremos el caso de la gestión de QoS en los servidores.

El control de tráfico en **Linux**, sinónimo en este caso de calidad de servicio, es el nombre dado a un grupo de sistemas de colas y mecanismos a partir de los cuales se define la transmisión de los datos en el sistema. Es decir, se incluyen en un computador capacidades para decidir el descarte, la priorización, el ratio de transmisión de los paquetes/tramas a recibir/enviar.

Dicha tecnología de QoS no solo permitirá establecer priorizaciones entre el tráfico generado entre las diferentes máquinas virtuales, sino que también ayudará a homogeneizar las latencias de red para que los diferentes procesos paralelos de los programas HPC tengan una carga de tiempo balanceada. Las soluciones típicas que nos ofrecerá el control de tráfico serán:

- Limitar y reservar el ancho de banda total a un ratio conocido.
- Limitar y reservar el ancho de banda para una aplicación/cliente/servicio determinado.
- Maximizar el throughput TCP en un enlace asimétrico; priorizar la transmisión de ACKs.
- Gestionar el tráfico sensible a la latencia
- Gestionar la sobresuscripción de tráfico
- Permitir el balanceo equitativo del ancho de banda no utilizado.
- Asegurar que un determinado tipo de tráfico no sea descartado.

Pasamos a describir los mecanismos tradicionales en el ámbito del control de tráfico explicadas con mayor detalle en el documento *"Traffic Control HOWTO"* ([04] Brown, 2006).

- ***Shaping:*** Mecanismo por el cual los paquetes se retrasan antes de la transmisión en una cola de salida para satisfacer una velocidad de salida deseada. Será interesante para homogeneizar los retardos existentes entre los diferentes puntos de los programas paralelos.

- ***Scheduling:*** Estrategia de planificación por la cual se ordena la salida de los paquetes en una cola en particular. Existen múltiples algoritmos para calcular el siguiente paquete a atender en una cola, lo cual permitirá diferenciar entre diferentes tipos de disciplina de cola:
  - *FIFO (first in – first out):* es el tipo de disciplina de cola más simple. El primer paquete en entrar en la cola, será el primero en ser salir de ella.
  - *PFIFO FAST*: Está basado en una disciplina de cola FIFO convencional, con la salvedad de que en este caso tiene tres colas FIFOs individuales para separar el diferente tráfico según la prioridad de este. Esta es la qdisc por defecto en sistemas tipo Linux.

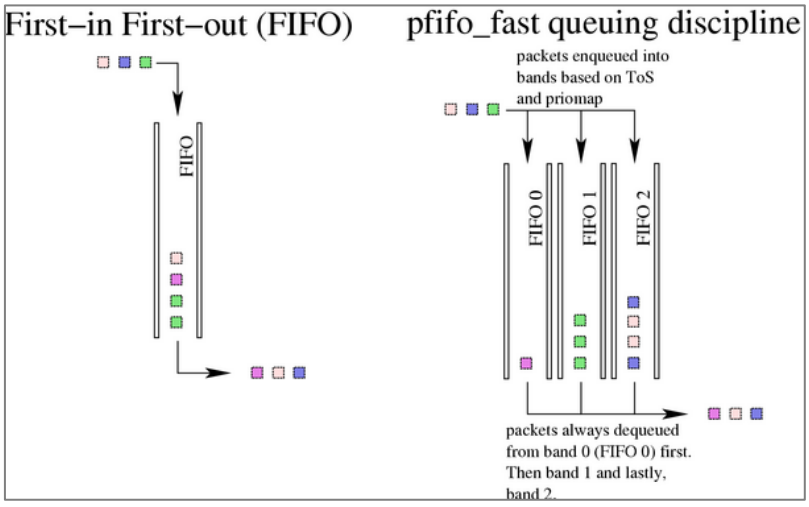

Ilustración 13 - Disciplinas de colas FIFO y PFIFO FAST

  - *SQF (stochastic fair queuing):* Este tipo de disciplina de cola intenta distribuir equitativamente la salida de los paquetes a un número arbitrario de flujos diferentes. Esto se logrará mediante una función *hash* que permite separar el tráfico en distintas colas FIFO, tras lo que se irán cursando paquetes de las diferentes colas mediante una estrategia *round-robin* (circular).
  - *ESFQ (extended stochastic fair queuing):* Es muy similar a SFQ pero con la diferencia de que se permite al usuario controlar qué algoritmo *hash* utilizar para distribuir el acceso a la red, y así se pueda llegar a una distribución más equitativa de ancho de banda.
  - *TBF (token bucket filter):* En esta estrategia de planificación de cola se configura el tráfico de transmisión en una interfaz, siendo la opción perfecta cuando se desee limitar la velocidad de los paquetes a ser quitados de una determinada cola. Se basa en *tokens* (ratio de tráfico deseado) y *buckets* (número de tokens), con lo que solo se retransmitirán datos de una cola cuando exista el número suficiente de *tokens*. De lo contrario serán diferidos, añadiendo latencia artificial a las comunicaciones de dicho tráfico.

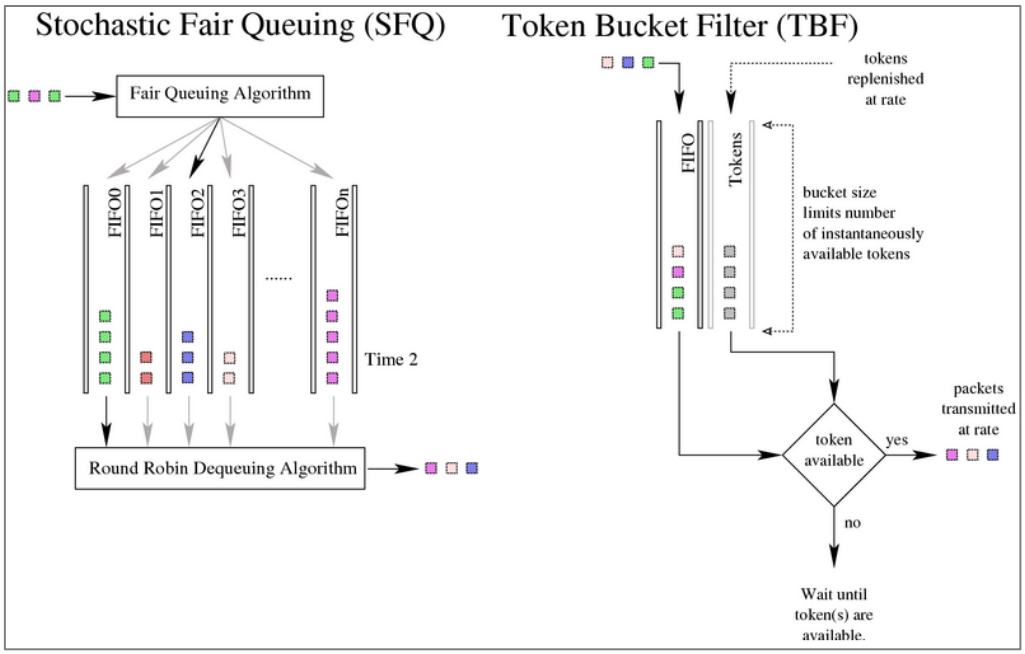

Ilustración 14 - Disciplinas de colas SFQ y TBF

- ***Classifying*:** Es un filtro mediante el cual los paquetes son clasificados para su posterior tratamiento diferenciado, todo ello posibilitado por la implementación de diferentes colas de salida. Esta clasificación puede incluir el marcado de paquete y nos permitirá tipificar los diferentes flujos de tráfico para su procesado diferenciado en los diferentes puntos de la red.
- ***Policing:*** Es un elemento de tipo filtro por el cual se puede limitar el tráfico tanto de entrada como de salida. Normalmente se suele utilizar en la zona frontera de la red. Es interesante para el estudio debido a que permite limitar la cantidad máxima de ancho de banda disponible para tráfico HPC, permitiendo tener una cantidad siempre disponible para este.
- ***Dropping:*** Es un mecanismo que permite el descarte de paquetes según ciertas condiciones preestablecidas.
- ***Marking:*** El marcado de tráfico es necesario para su identificación en los diferentes puntos de la red con el fin de poder tratar de forma diferenciada los diferentes flujos.

4.5.3 QoS en conmutadores

En este apartado se comentarán los aspectos relativos a QoS en conmutadores basándonos en la información del documento "*Enabling class of services on EX series switches in a campus LAN*" ([23], 2010).

Para una aplicación correcta de QoS deberemos tener en cuenta:

- En primer lugar deben agruparse y tratarse de la misma manera tráficos que tienen similares características;
- En segundo lugar deben conocerse los diferentes planificadores que describirán la forma de despachar las colas de salida asociadas a un único puerto;
- Finalmente debemos tener presente las diferentes formas de las que dispone el conmutador para gestionar los búferes cuando estos se saturan.

Para ofrecer una QoS una la infraestructura de red debe de presentar las siguientes características:

- Las colas implementadas en los búferes deben tener la suficiente granularidad como para albergar diferentes tipos de tráfico. Para un mejor rendimiento de las comunicaciones, los

diferentes tipos de tráfico han de ser gestiondos de forma diferente. Los conmutadores han de ser capaces de soportar varias colas diferentes por puerto con el objetivo de poder dedicar cada una a tipos diferentes de tráfico. *Las políticas de calidad de servicio deben ser consistentes en todos los dipositivos de red.* El objetivo de la calidad de servicio es ofrecer una manera de predecir el rendimiento de las diferentes comunicaciones que atraviesen los dispositivos de red; por lo que esto no será posible si existen elementos intermedios que no dispongan de las mismas capacidades de QoS. Es decir, de nada sirve aplicar una priorización del tráfico HPC en un solo conmutador de nuestra infraestructura, si en el resto de los dispositivos no las aplicamos.

- *Se debe tener la capacidad de gestionar, configurar y controlar la calidad de servicio de la infraestructura.* La utilización de herramientas software para la configuración y monitorización de los diferentes parámetros de la red de una organización, no solo ayuda a controlar si se está dando la calidad de servicio necesario; sino que también facilita su mantenimiento.

A continuación se muestra en orden el flujo de mecanismos de QoS que exprimentará un paquete desde que entra en el conmutador hasta que sale de este.

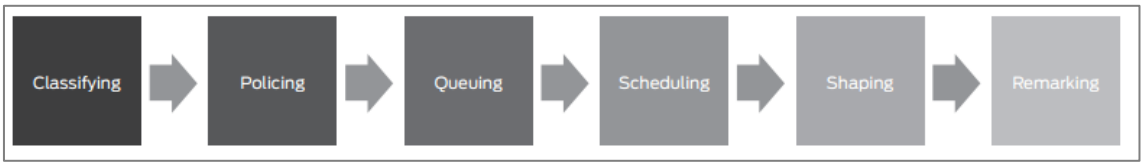

Ilustración 15 - Flujo de un paquete/trama en el conmutador

- ***Classifying*:** Se diferenciará el tráfico basándose en la información de las capas L2/L3/L4 que se obtiene del paquete/frame.
  La clasificación de tráfico es muy útil, principalmente debido a que permitirá el trato diferenciado de los diferentes tipos de este. En la siguiente figura podemos ver un esquema típico de colas virtuales de un puerto de salida organizadas según el tipo de tráfico que llegue a ellas.

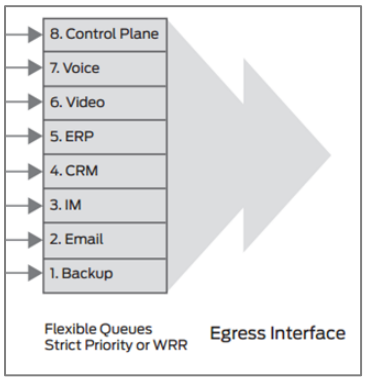

Ilustración 16 - Colas de conmutador según tipo de tráfico (clasificación)

- ***Policing*:** El control de adminisión determina que tráfico tiene permitido la entrada en el conmutador. Estos determinan si un flujo está utilizando el ratio de tráfico que se le ha asignado; o bien el comienzo de rechazo de paquetes debido a que se ha violado este límite. Si hay exceso de capacidad de la red disponible, puede marcarlos para su posterior manipulación en la red.
- ***Queuing:*** El encolado de los paquetes/frames es uno de los mecanismo claves de la QoS. Se habilitará el encolado para algún puerto que potencialmente presente peligro de

congestión. La granularidad del encolado vendrá dada tanto por el número de colas que puede soportar cada puerto, como por los diferentes tipos de algoritmos utilizados para mover paquetes desde las colas al puerto de salida. Las redes LAN típicamente soportan 2, 4 o 8 colas por puerto.

- ***Scheduling*:** Los algoritmos de *scheduling* permiten tomar decisiones de cuál de las múltiples cola debe atenderse primero para llevar el paquete/trama al puerto de salida. Ejemplos de estos mecanismos son:

  - *Strict-Priority (SP)*: En este algoritmo se irá despachando las colas según la prioridad de su servicio de forma secuencial.
  - *Shaped deficit weighted round-robin (SDWRR)*: En este caso se irán rotando los flujos a atender, emitiendo más paquetes/tramas al puerto de salida los pertenecientes a las colas prioritarias, pero intercalándolos con la emisión de flujos de otras prioridades más bajas.

- ***Shaping*:** Los mecanismos de *shaping* permiten conformar el tráfico cambiando sus propiedades de tasa de pico, tasa media u otros.

Los mecanismos de gestión de la congestión permiten controlar los flujos de tráfico. Ejemplos de estos son:

- *Weighted tail drop (WTD):* Cuando las colas alcanzan cierto límite de su capacidad se impide la entrada en la cola de paquetes/frames marcados como descartables.
- *Weighted random early detection (WRED):* Paquetes aleatorios con un se descartan cuando la cola llega a cierto límite de su capacidad.

5 Estudio de estado del arte

A continuación se analizará una serie de estudios en los diversos ámbitos que afectan a nuestro escenario teniendo en cuenta las diversas claves del mismo: Compartición de infraestructura, baja latencia y virtualización.

Artículo 1: "Less is More: Trading a Little Bandwidth for UItra Low Latency in the Data Center" ([01] Alizadeh, Kabbani, Edsall, Prabhakar, & Vahdat, 2012)

Este artículo, elaborado por CISCO, Google y la universidad de Standford enfoca el problema de la latencia en entornos de LAN proponiendo una arquitectura denominada ***HULL*** (*Highbandwidth Ultra-Low Latency*).

En este artículo se comenta la casuística de la existencia de recursos compartidos para flujos de tráfico HPC y de otros usos de tipo TCP concluyendo que este último intentará utilizar lo máximo los búferes del conmutador disponibles con el objetivo de aumentar el ancho de banda, a costa aumentar la latencia en consecuencia. Proponen la arquitectura HULL para eliminar/reducir la utilización de búferes en el conmutador por el tráfico convencional a partir de un marcado de la congestión basado en el aprovechamiento de la utilización del enlace, dejando las colas libres para tráfico sensible a latencia como el HPC. Es decir, limitarán la cantidad de ancho de banda disponible de un enlace a cambio de una reducción significativa de latencia.

La implementación de *HULL* está basada en *Phantom Queues*, un mecanismo del conmutador relacionado con la existencia de un planificador de colas basadas en colas virtuales. Éste simula la ocupación de una cola vinculada a un enlace (en un puerto del conmutador), en las que se marcará el tráfico basándose en la utilización del enlace en vez de a partir de la ocupación de la cola con el estándar *ECB* (una extensión del protocolo IP y TCP que permitirá notificar congestión en la red cuando exista pérdida de paquetes).

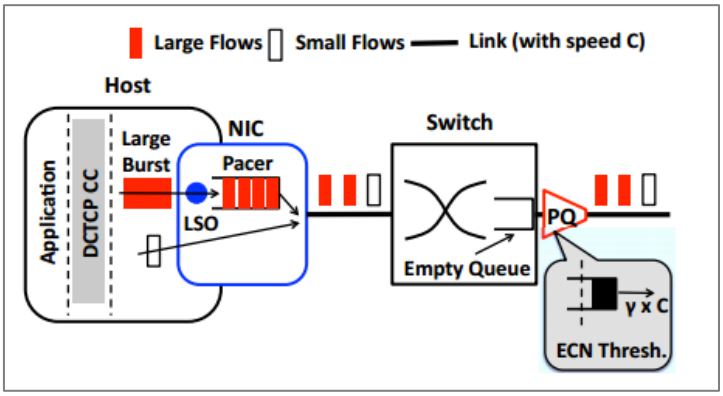

Ilustración 17 - Esquema de la arquitectura HULL

Las Phantom Queues no serán realmente una cola debido a que no almacena paquetes en ella, sino que simplemente es un contador que se actualiza a medida que los paquetes salen por el enlace a velocidad de la línea, determinando porque cola virtual deberían haber entrado. Tras esto el estándar ECN marcará la ocupación de las Phantom Queues, tras lo que se utilizará

en los host finales el algoritmo de congestión DCTCP (*Data Center TCP*) para reducir la tasa de retransmisión si existe congestión. En definitiva, las *Phantom Queues* intentan establecer tasas de transmisión de flujos estrictamente menores que la capacidad del enlace, manteniendo así una cantidad considerable de grandes búferes desocupados que podrán ser utilizados por aplicaciones sensibles a latencia.

También concluyen que es necesario utilizar *"hardware packet pacing"* (es un mecanismo que inserta espacio adecuado entre los paquetes de un flujo transmitidos por los hosts) para suavizar la contraproducente transmisión en ráfagas de tráfico que producen mecanismos de estimulación como *Large Send Offloading* e *Interrupt Coalescing*. El objetivo es introducir métodos innovadores para estimar una tasa de transmisión que sea amigable con la congestión mediante mecanismo de espaciado. Sin éste, las *Phantom queues* probablemente se confundirían al marcar congestión cuando le llegasen grandes ráfagas de transmisión que producen algunas técnicas de *"offloading"*, causando degradación en el ancho de banda. En principio, este espaciado no es necesario para el tráfico sensible a latencia.

En su experimento utilizarán el hardware *NetFPGA*, el cual se conectará en un punto intermedio entre los conmutadores y los servidores, y donde también se implementarán las colas virtuales, las *Phantom queues* y los mecanismos de *"hardware packet pacing"*. Finalmente se realizan una serie de pruebas entre pares de diez máquinas, y obtienen que su arquitectura HULL es mejor en el percentil 90th de las medidas de latencia realizadas, pero peor TCP con QoS en el percentil 99th. Generalmente, cuando el tamaño de *buffer* en el conmutador es grande, se comporta mejor el protocolo TCP con técnicas de QoS; pero en cambio, cuando los búferes de los conmutadores son más pequeños, se comporta mejor la arquitectura HULL.

Artículo 2: "Towards Virtual Infiniband Clusters with Networks and Performance Isolation" ([14] Hillenbrand, 2011)

En este estudio se realiza una aproximación de como virtualizar un *clúster* con infraestructura de alto rendimiento (en concreto con una red de interconexión *Infiniband*). Se propone que un usuario pueda crear sus propios clústeres locales mediante la reserva de recursos del clúster virtual global utilizando mecanismos de calidad de servicio de Infiniband de aislamiento de red. También proponen la ejecución tráficos no HPC en los ciclos que se pierden entre ejecución y ejecución de trabajos de alto rendimiento.

Utilizan la tecnología virtualización KVM, y para solucionar el problema de la comunicación entre los domU y la tarjeta de red (recordemos que estos accederán a los recursos físicos en modo no privilegiado) – en el caso Infiniband al acceso del host channel adapter (HCAs) – utilizarán la técnica llamada *VMM-bypass I/O*. Esto es una implementación de los drivers virtuales que permite a un proceso del domU emitir directamente las peticiones de comunicación al Infiniband HCA, conservando el aislamiento entre las máquinas virtuales y entre los procesos.

En cuanto a la ejecución de trabajos no HPC se propone la utilización de las máquinas virtuales cuando estén ociosas (es decir entre el tiempo después de acabar un trabajo, y el tiempo antes de comenzar). Por lo tanto se necesitará un planificador para mandar el trabajo no HPC a otro lado cuando un nuevo trabajo HPC llegue. Para gestionar esta prioridad trabajan con el componente *Linux-specific SCHED_IDLE scheduling policy* (es compatible con Infiniband).

En la parte de red interesa generar las topologías que demande el usuario: además de que cada entorno de trabajo esté aislado tanto del exterior, como de los trabajos de los otros usuarios. Para ello se utilizarán técnicas de aislamiento, QoS y *policy*.

Implementan el aislamiento entre tráficos de red de diferentes clústeres virtuales empleando la técnica de particiones de *Infiniband* (se puede dividir la red en subredes y restringir su comunicación con otros nodos). Se comparte ancho de banda entre varios clúster virtuales que comparten un enlace físico y se establece límites en el retardo de los paquetes experimentado debido a la presencia de contención (usando los mecanismos de calidad de servicio de Infiniband). Se emplearán los mismos mecanismos para utilizar una parte de los recursos de la red para la gestión del Cloud y el tráfico que no es HPC. Se puede particionar recursivamente la red de un clúster virtual creado por un usuario.

Después también se define el mecanismo de calidad de servicio de Infiniband, que permite separar flujos de red y planificar el ancho de banda (se utilizará esto para la compartición de recursos entre los diferentes flujos existentes en la infraestructura física). Infiniband para diferenciar flujos utiliza 16 niveles de servicios diferentes (SL) y ofrece sus 16 colas de salida por puerto (VL) que se denominan "*virtual Lanes*" y que arbitrarán el ancho de banda físico.

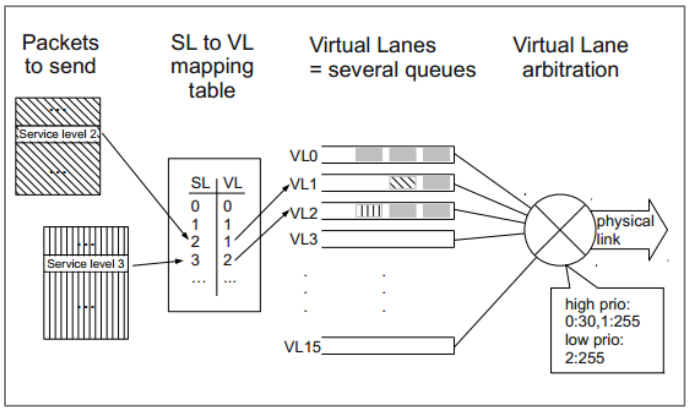

Ilustración 18 - Mecanismo de QoS de Infiniband

Como herramienta de gestión del *cloud* se utilizará el framework OpenNebula (reserva de recursos, creación del clúster virtual, etc).

En las conclusiones, se anuncia que su estructura de clúster virtual sería adecuada para trabajos paralelos pequeños y medianos. También encuentran que la utilización de los tiempos ociosos de las CPUs para trabajos no HPC (posible gracias al planificador *policy* de Linux) causan una mínima pérdida de rendimiento en la degradación de aplicaciones de uso intensivo de CPU. En cambio para los trabajos de uso extensivo de memoria si pueden verse afectadas por las máquinas que estén realizando trabajos de baja prioridad que se ejecuten en otros núcleos de la máquina física. Esto demuestra que los planificadores *policy* del sistema operativo no tienen influencia a la hora de arbitrar recursos hardware como los buses de memoria compartida del servidor.

Artículo 3: "EyeQ: Practical Network Performance Isolation for Multi-tenant Cloud" ([18] Jeyakumar, Alizadeh, Màzieres, Prabhakar, & Kim, 2012)

Este tercer estudio fue realizado por la *Universidad de Standford* y *Windows Azure*. En primer lugar describe los problemas de la compartición de recursos de red entre los múltiples tráficos que existirán en la organización: la congestión puede aparecer en cualquier punto de la red, dificultades en la diferenciación del tráfico por los requisitos que necesita cada uno; o el fenómeno de ráfagas de tráfico que presentan protocolos como TCP y UDP (lo cual puede congestionar nuestro conmutador en un momento inesperado). Plantearán sus optimizaciones

partiendo del punto de vista de tomar la topología de red del centro de datos como una matriz de conmutación gigante, en la que los puertos de entrada y de salida serán hosts emisores y receptores. Reservarán un poco del ancho de banda de los enlaces de la infraestructura (10%), para obtener la aceleración típica que tienen los conmutadores en los puertos de salida para reducir los problemas HOL en los puertos de entrada, y así emular lar colas de los puertos de un conmutador.

Por lo tanto, los problemas de congestión y contención de paquetes, tal y como ocurre en un conmutador real, no estarán en la matriz de conmutación (topología de red de la organización en este caso), sino que estarán en la cola de salida o bien en el ratio de transmisión del emisor. Por ello desplazan los problemas de red a los bordes de la infraestructura, proponiendo el sistema *EyeQ*, a partir del cual mediante la utilización de mecanismos de planificación de paquetes en los hosts de emisión y recepción, se pueden establecer políticas de límite de velocidad y control de congestión, para que garanticen una buena calidad de servicio en las comunicaciones de la organización.

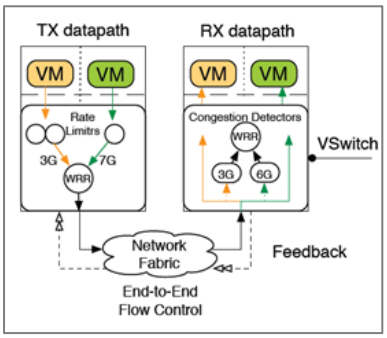

Ilustración 19 – Solución propuesta por EyeQ

Como se puede ver en la figura, el componen TX consistirá en un *policy* que limite la tasa de transferencia utilizando un planificador WRR (en RoundRobin) que garantizará a cada VM un mínimo de ancho de banda (será análogo a la técnica de colas virtuales que se ha hablado en el capítulo 2 - VOQ). Por otro lado, el componente RX consiste en un número de detectores de congestión, uno por máquina virtual, el cual también tendrá asociado un planificador WRR con el que asigna un ratio de transferencia a cada VM. El control de congestión se cerrará ante la llegada de paquetes si una máquina excede este ratio de tráfico, enviando al receptor un mensaje de 1 solo bit (como el protocolo ECN) para que limite su ratio de transferencia debido a que existe congestión.

Artículo 4: "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud" ([16] Jackson, y otros, 2010)

En este artículo se experimenta sobre las posibilidades del *clúster* de computación de altas prestaciones de Amazon. Se hará una comparativa entre los servicios del clúster virtual que ofrece Amazon (el sistema EC), con otros clúster y superordenadores físicos mediante una aplicación benchmark real (como GAMESS, MAESTRO, etc). El resultado es que el sistema virtualizado EC es un orden de 6 veces más lento que un clúster genérico Linux; y hasta 20 veces más que un superordenador. Esto principalmente es debido a que Amazon EC2 tendrá sus nodos en la misma región (en el caso del estudio en *US East*), pero no se ofrecen garantías de que exista proximidad en los nodos que se asignan, por lo que existe variabilidad significativa en las latencias entre nodo (y como ya se sabe, en las aplicaciones paralelas el nodo más lento es el que determinará el tiempo de ejecución de la aplicación). Además de eso, también describen

el problema de que los nodos de sus instancias virtuales pueden estar compartiendo recursos con otros tráficos que están haciendo uso tanto de los recursos del servidor (memoria, bus del sistema, tarjeta de red) como de la parte de red; y que seguramente exista degradación en sus pruebas por ese motivo.

Artículo 5: "Congestion Management and Buffering in Data Center Networks" ([12], 2012)

En este *whitepaper* no se habla exactamente de la realización de HPC en entornos *cloud* pero sí trata el problema de múltiples flujos de trabajos en las redes de los centros de datos. Como ya se ha dicho, los protocolos como TCP tienen un mecanismo incorporado para la gestión de la congestión, debido al número de saltos entre origen y destino. El tiempo que se tarda en llevar la información de congestión a la pila TCP en el origen puede ser lo suficientemente grande como para que sea necesario un gran tamaño de buffer en los puntos de congestión.

*B (tamaño de buffer) = C (data rate of the link) * RTT*

Esto puede ser cierto cuando se trata de un único flujo de datos. Pero la realidad es que los conmutadores de los Cloud organizativos, normalmente sirven a un número elevados de flujos simultáneos (de larga o corta duración). Su investigación llega a que cuanto existe una mezcla de flujos, tanto de corta duración, como de larga, para un número "n" de estos se requerirá un tamaño de buffer no mayor de:

*B = (C * RTT)/raíz(N)*

El problema es que recientemente se han mostrado teorías en las que se dice que utilizar búferes demasiado grandes en una red de conmutación de paquetes provoca una alta latencia y jitter, así como una reducción del rendimiento global de la red. A este fenómeno se le llama "***Bufferbloat***", y fue propuesto por Jim Gettys en 2009. Como ya se ha comentado, se sabe que el algoritmo de congestión TCP saturará el buffer del conmutador para determinar el ancho de banda disponible, y ajustará la velocidad de transmisión una vez los paquetes comiencen a ser descartados. Si tenemos un *buffer* demasiado grande que ha sido saturado, los paquetes llegarán a su destino pero con una mayor latencia. Esto es debido que a mayor tamaño de esta memoria, más se tardará en descartar paquetes, debido a que la ventana de congestión de TCP solo se redimensionará cuando el buffer esté totalmente lleno. El problema es que contextos de un RTT muy pequeño como en una red local, quizás el paquete pase más tiempo en la cola del conmutador, de el que se tardaría en ser reenviado y ser atendido en una cola más pequeña.

Por lo tanto, el estudio afirma que la presencia de búferes demasiado grandes en los conmutadores retiene a los paquetes en la cola demasiado tiempo, causando que TCP responda mucho más lento y tal vez demasiado tarde a la congestión en la red. Esto tiene el efecto de abrir la ventana de congestión, sobreestimar incorrectamente el tamaño del pipe y conducir más tráfico desde el origen, especialmente en un momento en que la red ya está congestionada.

Con el fin de combatir este y otros problemas como la naturaleza de ráfagas de tráfico de Ethernet, proponen dos principales consideraciones a la hora de analizar un switch:

- ***Buffer allocation strategy :*** se refiere a la forma de reservar memoria para la cola de cada puerto del conmutador. Una estrategia dinámica permitiría utilizar un "*pool*" de búferes compartidos que utilicen los puertos en momentos de congestión. Esto ofrece la flexibilidad de ofrecer almacenamiento de tamaño dinámico según las características del tráfico a almacenar.

- ***Optimal buffer size :*** es decir, que es importante conocer el tamaño óptimo de buffer que se necesita en un caso concreto según el tráfico que vayamos a transportar. El tamaño ha de ser lo mínimamente grande como para almacenar las ráfagas de tráfico de algunas aplicaciones, y así impedir congestión en momentos inesperados.

Para la implementación de sus conmutadores proponen la tecnología *smart buffer* implementada por BroadCom y explicada en el artículo "*Broadcom Smart-Buffer Technology in Data Center Switches for Cost-Effective Performance Scaling of Cloud Applications"* ([08] Das & Sankar, 2012) ([08] Das & Sankar, 2012) ([08] Das & Sankar, 2012) en la que se dice que incorpora un diseño escalable con una arquitectura de memoria centralizada. En este último documento se explica que los búferes del conmutador son del tamaño adecuado (no son excesivamente grandes) y que son compartidos de manera dinámica a través de todos los puertos para absorber ráfagas de tráfico, evitando así situaciones innecesarias de congestión. Adicionalmente se asegura que los puertos de salida utilizarán el pool compartido solo cuando sea necesario (detener la congestión, siempre y cuando nos interese tener un buffer más grande).

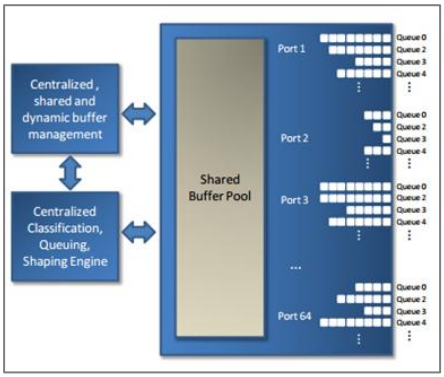

Ilustración 20 - Tecnología Smart-Buffer: gestión de buffers dinámica y centralizada

Como conclusión vinculada el caso de estudio de la competencia de tráficos HPC con otros en un Cloud organizativo, se extrae que la presencia de búferes de tamaño grande en la infraestructura local con un RTT muy pequeño, introducirá lag en las comunicaciones. Esto será perjudicial para la ejecución de tráfico sensible a latencia. Pero esto no quiere decir que el tamaño de buffer tiene que es muy pequeño: las características de tráfico a ráfagas de las redes Ethernet hace necesario que tengamos un tamaño de cola en los conmutadores como para tolerar estos fenómenos. De no ser así, las colas de los puertos de salida estarían en congestión continuamente, perjudicando a la calidad de todas las comunicaciones que pasen por el dispositivo.

Artículo 6: "SCTP versus TCP for MPI" ([26] Kamal, Penoff, & Wagner).

En este artículo se propone la utilización del protocolo de transporte SCTP, según indica este proporciona confiabilidad, control de flujo y secuenciación como TCP, pero está orientado a mensajes como UDP, lo cual permite el envío de estos fuera de orden. Su característica más importante es el denominado *multi-streaming*, con lo que se refiere a la capacidad de enviar en un mismo mensaje pedazos de información de varios flujos diferentes. Es decir, el protocolo agrupa todos sus flujos de información en uno solo que se mandará con fragmentos de todos. Esto es importante, debido a que para todas las comunicaciones con SCTP que salgan de un mismo servidor se eliminará la presencia de bloqueos HOL en los puertos de entrada de los

conmutadores y la contención en los puertos de salida, principalmente debido a que el tráfico de un mismo *host* no competirán entre sí debido a que son transmitidos en paralelo.

Artículo 7: "Design and Implementation of Open-MX: High-Performance Message Passing over generic Ethernet hardware" ([13] Goglin, 2012)

Open MX es una implementación de la pila de paso de mensajes *Myrinet Express* bajo redes genéricas ethernet. Este implementará una nueva capa en la parte superior de la pila *ethernet* de los *kernel* de Linux, que será interoperable con redes Myrinet. Utiliza técnicas como *kernel-bypass* y *zero-copy* para obtener unos tiempos bastante superiores a las comunicaciones con TCP (sobre todo si se usa 10Gbit Ethernet). La pila de este protocolo solo tendrá tres capas: la primera será la de aplicación (programas MPI junto con la librería en cuestión), después estará MXoE (*Myrinet Express over Ethernet*) que se corresponderá con la capa de red; y por último la capa física. Debido a que el mensaje trabaja principalmente en la capa de red y de aplicación, no tiene control de congestión como TCP.

Artículo 8: "High Performance Parallel Computing with Clouds and Cloud Technologies" ([11] Ekanayake & Fox, 2009)

Según conocemos la arquitectura de Xen Hypervisor los *domU* de una misma máquina no serán capaces de realizar operaciones de E/S por sí mismos, si no que deberán de comunicarse con el *dom0* (el cual cuenta con un sistema operativo privilegiado para el manejo de recursos) a través de un canal de eventos (interrupciones) y de la memoria compartida del procesador en sistemas *multicore*. Esto provoca que si optamos por una configuración en la que los procesos MPI se ejecutan en múltiples *domU* pertenecientes al mismo servidor físico, siempre tendrá que estar el *dom0* como intermediario de estas comunicaciones, añadiendo latencia y desaprovechando características como la caché compartida (como se puede ver en la segunda figura). Esto es debido a que el anfitrión es el único que tiene los *backend drivers* de disco, con lo que solo él accederá directamente al dispositivo (el resto lo hará a partir de él). Adicionalmente, cuando se desee establecer comunicaciones con el exterior, es decir utilizando ya la tarjeta de red, también se añadirá *overhead* debido a que el anfitrión no tiene privilegios (no puede acceder directamente a los recursos de red del sistema). Por lo tanto el *"guest"* tendrá también que comunicarse con la máquina "anfitrión", lo cual añadirá tiempos de latencia extra a tener en cuenta, debido a que el *dom0* también tiene los *backend driver* de la tarjeta de red, con lo que solo él accederá en modo privilegiado a esta.

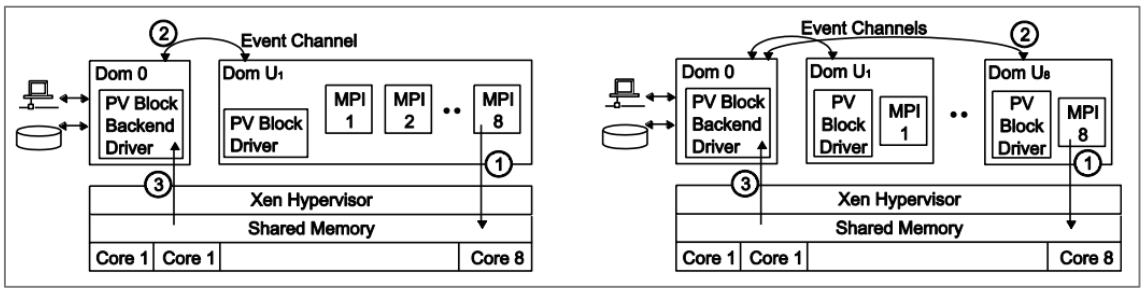

Ilustración 21 - Comunicaciones entre domU en arquitecturas para-virtuales

En el caso contrario, si tenemos pocas *domU* que ejecutan los procesos MPI sobre múltiples cores que tiene asignados a si misma (y que pertenecen también a una misma máquina física), las comunicaciones con el gestor de eventos del dom0 se reducirá mucho, y por lo tanto no se añadirá demasiada latencia (es el primer caso de la anterior figura). El problema de una

configuración de estas características es el acopio de recursos en las instancias y la pérdida de versatilidad.

En el caso de un *cloud* privado multipropósito, esto es un problema, debido a que deben disponerse siempre los recursos necesarios para abastecerlas. Además, no siempre se obtendrá unas menores latencias debido a que se requerirá que la implementación de MPI que estemos utilizando tenga soporte mejorado en los nodos de comunicación multi-core (por ejemplo LAM MPI no lo tenía).

Por lo tanto, tendremos que el dom0 puede convertirse en un cuello de botella debido a que en las comunicaciones que realicen los *domU* estarán accediendo constantemente a éste. El rendimiento de red del anfitrión se degradará, lo que perjudicará al servicio del resto de máquinas "*huéspedes*".

Artículo 9: "The TCP Outcast Problem: Exposing Unfairness in Data Center Networks" ([29] Prakash, Dixit, Charlie Hu, & Kompella, 2012)

En este estudio analizan el problema de la compartición de ancho de banda a través de la infraestructura de red de un centro de datos (normalmente ordenados por topologías tipo árbol) por múltiples flujos de tráfico TCP. La principal incidencia que estudian es el "*TCP Outcast*", el cual ocurrirá cuando en dos puertos de entrada llegan flujos de tráfico de diferentes orígenes. Por uno se transportarán muchos flujos, por otro pocos; y el destino de ambos es el mismo puerto de salida. Se observa que el pequeño conjunto de flujos perderá cuota de rendimiento significativamente con respecto al otro. El *TCP outcast* se presenta normalmente cuando el mecanismo de congestión de las colas de los puertos de salida de los conmutadores es *taildrop*.

En su investigación han detectado un curioso fenómeno conocido como *port blackout*, donde una serie de paquetes de uno pertenecientes a un mismo puerto de entrada serán desechados, mientras que otros cuyo origen es distinto puerto, y cuyo destino es el mismo puerto de salida serán conmutados perfectamente. Esto normalmente afectará más al caudal perteneciente al puerto de entrada que transporta poco flujo de datos. Para solucionar este problema ofrecen distintas soluciones:

- *RED* : Es una política activa de gestión de colas con la que se detecta la congestión incipiente y marca aleatoriamente los paquete para evitar la sincronización de la ventana. El marcado aleatorio de los paquetes entrelaza los paquetes de los diferentes puertos de entrada, lo que permite que cuando exista congestión y estos tengan que ser desechado, se ha de una manera equitativa. Por lo tanto, esto evita el efecto blackout port.
- *SFQ* : Utilizan el algoritmo *Stochastic Fair Queing* como planificador, para compartir el ancho de banda entre todos los flujos que llegan al conmutador.
- *TCP pacing :* también conocido como espaciado de paquete, es una técnica que ya se visto en la literatura, por la que se espacia los paquetes enviados asociándolos a una tasa de transmisión concreta. Esto también reduce significativamente el blackout port.
- *Equal-length :* Como en la investigación llegan a la conclusión de que una de las principales causas del problema Outcast TCP es las diferentes distancias que habrá entre remitentes y emisores, provocará que existan múltiples flujos compitiendo por los mismos puertos de salida constantemente. La solución que proponen con el objetivo de que todos los paquetes recorran caminos parecido, y homogeneizar así sus RTT, es hacer que los paquetes sean enviados al conmutador de la topología (esto es la raíz del árbol) para realizar cualquier transmisión.

Artículo 10: "Differentiated Network QoS in Xen" ([28] Mittal, 2009)

En este último artículo se dispone a crear un planificador para E/S de red, con el que se arbitre el uso de recursos de red entre los diferentes dominios huéspedes de Xen. Para ello se basara en la utilización del algoritmo de planificación *DWRR* (en Round Robin dinámico); y de la planificación *CPU Credit Scheduler* (que es el encargado de balancear el uso de las CPUs físicas con respecto a las VCPU – CPU virtuales – de cada domU).

El algoritmo de planificación se basa en darle una serie de créditos a las diferentes máquinas huéspedes. A medida que un dominio está transmitiendo paquetes, los créditos asignados se le irán disminuyendo (una forma de hacerlo es restar a estos créditos el tamaño de los paquetes que se van enviando). Una vez se le agoten estos, se llamará a una función de asignación para que se marque dicho dominio, se bloquee y se mueva a la cola de dominios bloqueados, a la espera de que se le den nuevos créditos para recomenzar la transmisión de datos.

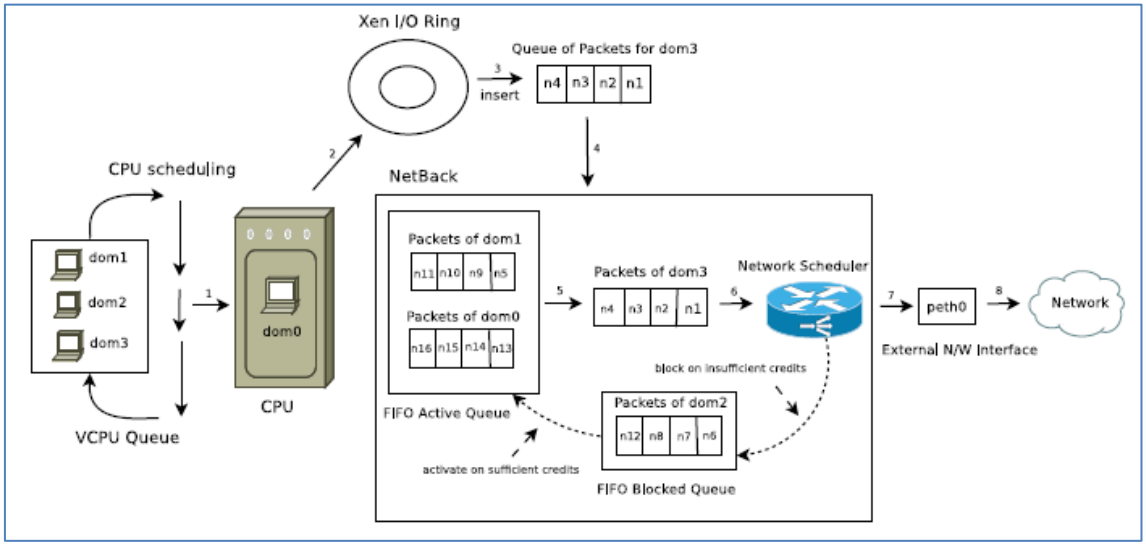

Ilustración 22 - Planificador Network I/O Credit Scheduler

Como resultado, se dispone de un buen mecanismo de balanceo de carga en el uso de los diferentes recursos de red, disminuyendo los posibles conflictos en el uso de recursos entre los tráficos de las diferentes máquinas virtuales huéspedes que estén desplegadas en un mismo servidor físico.

6 Propuesta de optimizaciones

De la información analizada y explicada a lo largo de los apartados anteriores pueden extraerse diferentes medidas o recomendaciones que se pueden acometer con el fin de mejorar las prestaciones de las aplicaciones HPC en entornos *cloud* multipropósito.

Dadas las restricciones temporales del presente trabajo no se considera objeto del mismo el análisis de su impacto real (ni es posible cuantificarlo) sino la propuesta de diversas medidas a acometer con el fin de evaluar su posible aplicación en dichos entornos.

6.1 Conmutadores

Las arquitecturas de los conmutadores de red son diversas según el sector de mercado al que se orientan y según su uso dentro de la LAN (acceso, distribución, core).

La arquitectura concreta de los conmutadores determinará los posibles lugares de contención y también puede influir en el momento en que ésta aparecerá (determinada por ejemplo según el tamaño de los búferes de almacenamiento). El conocimiento de esta arquitectura puede guiarnos para determinar la distribución óptima de los servidores a los puertos de los conmutadores con el fin de optimizar la latencia de los flujos de información en función de la conectividad interna del conmutador (por ej., en caso de *clusters* de tamaño reducido podemos favorecer la conexión de los servidores anfitrión a puertos de un conmutador pertenecientes a la misma PFE o del mismo miembro de un chasis virtual).

Adicionalmente, los diversos fabricantes optan por versiones mejoradas de conmutadores *ethernet* diseñados con orientación a la captación de mercado HPC teniendo en cuenta su necesidad de baja latencia y *jitter*. Un conmutador 10 *gigabit ethernet* orientado a aplicaciones HPC puede conmutar tramas de 1518 bytes, según datos de fabricante, con una latencia media de 0.86µs (0.97 µs máx)[3] en el caso del Juniper SFX3500 o bien los 300ns en el caso de los equipos del fabricante FORCE 10 (ahora DELL)[4]; en contraposición están los 13.46µs del Juniper EX 8200 en configuración de chasis virtual[5]. La tecnología de acceso es también determinante, ya que la latencia de conmutar una trama de 1518 bytes por puertos gigabit es de 26,24µs en el caso de los equipos EX4200 del mismo fabricante[6].

Según los valores de retardo indicados parece aconsejable minimizar el número de conmutadores que atraviesan los flujos de HPC para su comunicación entre miembros del clúster dentro del *datacenter*. La utilización de tecnologías 10 *gigabit* de ultra baja latencia favorecerá el despliegue de servicios HPC en *ethernet*. El diseño de red realizado, en relación a las interconexiones de los conmutadores/*routers* en la LAN, requiere también una revisión detallada de cara a minimizar el retardo e igualar los *paths* que sigue el tráfico entre los diferentes pares de *domU* de un *cluster* HPC. Las arquitecturas de red convencionales pueden implicar caminos poco óptimos en este tipo de escenarios donde se requieren caminos entre anfitriones con latencias bajas y de poco *jitter* (caminos similares).

[3] Información suministrada por el fabricante de conmutadores Juniper ([25], 2011).
[4] Información suministrada por el fabricante de conmutadores DELL ([10], 2006)
[5] Información suministrada por el fabricante de conmutadores Juniper ([24], 2012)
[6] Información suministrada por la revista Network World ([21], 2008)

6.2 Aplicaciones y protocolos de comunicaciones

De análisis de las aplicaciones y protocolos de red existen diversas mejoras que cabe ponderar:

- Utilización de mecanismos *kernel-bypass* y *zero copy*.
- Utilización de protocolos alternativos a TCP como mecanismo de transporte, un ejemplo es Open MX, que trabaja directamente sobre ethernet y dispone de mecanismos de *kernel bypass* y *zero copy* en sí mismo.
- Evaluación de diferentes algoritmos de control de congestión TCP, en nuestro escenario dadas sus características cabría evaluar las mejoras aportadas por CUBIC y DCTCP.
- Utilización de otros protocolos de transporte como el *SCTP*, que opera directamente sobre UDP y elimina el riesgo de bloqueo HOL

6.3 Tecnologías de virtualización

Según analizamos en el apartado de tecnologías de virtualización con el caso del hipervisor Xen y, a raíz del artículo 8: *"High Performance Parallel Computing with Clouds and Cloud Technologies"* ([11] Ekanayake & Fox, 2009) podemos indicar que, ante ciertos patrones de comunicación entre domU de una misma máquina anfitriona el dom0 puede convertirse en un cuello de botella por ser quien gestiona en todo momento la interacción con los dispositivos mediante los *backend drivers*.

Existen optimizaciones como la propuesta de Xen de crear un ***driver domain*** con el objetivo de eliminar este cuello de botella. Se tratará de una instancia virtual que tiene la responsabilidad de gestionar los accesos no privilegiados a un determinado componente *hardware* (en este caso la tarjeta de red). En él se ejecutará un *kernel* mínimo que solo contendrá el driver real del dispositivo que controla (y que tiene acceso a éste) y los *backend drivers* (uno por cada *guest*), que serán los controladores que permitan a los *domU* el acceso a la tarjeta de red.

En el caso del hipervisor Xen, el soporte de dispositivos se basa en la división de los drivers en dos partes. En los *domU* existirá un driver de red, que se denomina *front-end driver* y el cual es el encargado de comunicarse con el *back-end driver* para la transmisión de datos, el cual es gestionado por el *dom0*. Cuando *domU* desea transferir un paquete de datos enviará una interferencia al hipervisor para notificar que la página será recibida por otros dominios. Tras esto, el dominio local se transfiere mediante la técnica *page-flipping*, la cual consiste en un buffer de memoria estático y circular compartido por los dos dominios (emisor y receptor), donde la información es escrita por uno de ellos, y leída asíncronamente por el otro.

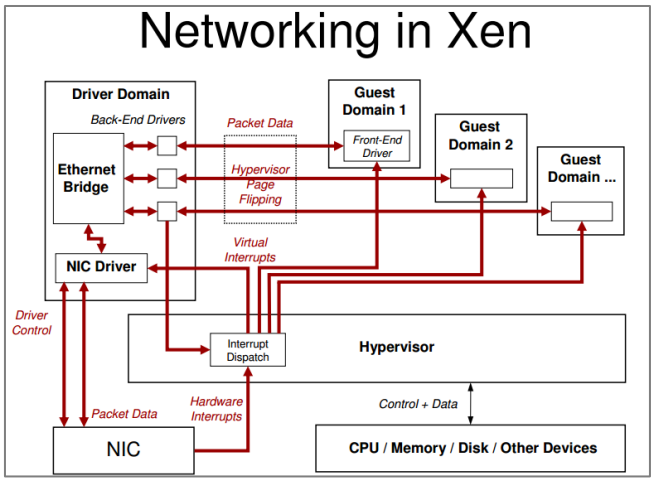

Ilustración 23 - Configuración de XEN mediante la creación de un Driver Domain

Unido a la utilización de esta técnica se puede utilizar la tecnología *PCI passthrough*, que permitirá dar el control de un dispositivo PCI a los *guest*, incluido el DMA La principal ventaja de esto es que se elimina el *overhead* de la virtualización en las comunicaciones de red; pero tiene los inconveniente de que puede ser potencialmente peligroso, ya que elimina el aislamiento de las máquina virtuales con respecto a la infraestructura física, puede ser inestable, y también reduce la flexibilidad de la migración de los *domU*, debido a que cuando se desee migrar un dispositivo a otro lugar, este debe tener disponible el modo *PCI passthrough*. En *full-virtualized*, se requiere hardware o chipset como IOMMU (Intel VT -d); en cambio en la paravirtualización, el *PCI passthrough* ha estado disponible durante años, y no requiere de ningún dispositivo específico para su utilización.

Sin estas optimizaciones, las aplicaciones MPI experimentan una degradación de rendimiento en la infraestructura virtualizada. Esto es un punto crucial a tratar, sobretodo en aplicaciones sensibles a latencias (el ancho de banda no se resiente demasiado).

En este trabajo no se tienen en cuenta la degradación que puede producir la virtualización al existir la compartición de otro tipo de elementos (disco, cpu, memorias,…).

Por último comentar que, del estudio de la arquitectura interna de Xen se observa que las solicitudes de entrada/salida de red realizadas por los *domU* se sirven de forma secuencial según una disciplina FIFO. No existe a día de hoy mecanismos que permitan priorizar la gestión de unas frente a otras del tráfico de red.

6.4 Linux y los dispositivos de red

Del análisis del funcionamiento de las tarjetas de red se observa que también nos condiciona el comportamiento del tráfico y existen ciertas mejoras que podremos llevar a cabo:

- La NIC de un servidor físico perteneciente al *cloud* privado será atravesada por diferentes tráficos de las máquinas virtuales a las que este atiende, siendo algunos de estos flujos prioritarios. En este caso tendremos un problema debido a que el sistema operativo irá creando y almacenando los descriptores de buffer de las tramas según estas vayan apareciendo. Esto es un problema si retenemos encolados a los descriptores de fichero de búferes correspondientes a tramas de tráfico prioritario. Por ello es evidente que se necesitan mecanismos de QoS en la parte servidor, mediante los cuales se podrá clasificar y priorizar la salida del tráfico del servidor físico.

- Ya hemos comentado la existencia del mecanismo de *checksum*; es posible deshabilitar la verificación *checksum* en el *domU*, siempre y cuando esta comprobación ya se haya realizado en los drivers de red del *dom0* (o bien en el *domain driver*), eliminando la sobrecarga que esto puede producir.
- Los mecanismos *large send offloading/TCP segmentation offload* (*tso*) pueden producir distorsiones en la distribución del tráfico. Cuando estos mecanismos están habilitados la tasa de envío de paquetes será mayor, con lo que provocarán ráfagas de datos que podrían congestionar los búferes de los conmutadores en cierta medida.
- En el caso del mecanismo de *coalescing*, al igual que el caso de tso también pueden producirse ráfagas que podrían congestionar la infraestructura de red más de lo normal, degradando los tráficos de la organización. Adicionalmente otro efecto adverso es la acumulación de tramas en recepción con el fin de evitar interrupciones al sistema operativo, lo que perjudica los valores de latencia.

6.5 Calidad de servicio

Es necesaria la aplicación de mecanismos de QoS en servidores y en conmutadores con el fin de priorizar el tráfico HPC en todos los elementos de red atravesados por el tráfico sensible a retardo.

El patrón de tráfico de los flujos no HPC influirá también en los parámetros de retardo. Cuando éste presenta ráfagas con patrones abruptos afectarán en gran medida al tráfico HPC, que deberá esperar a ser servido en los diferentes búferes de la red. Esta situación se beneficiará de la aplicación de políticas de QoS tradicionales extremo a extremo, eliminando tiempos de espera en búferes intermedios.

Aun cuando se aplica QoS los paquetes prioritarios sufren retardo debido a paquetes no HPC que se estén cursando. Por este mismo motivo se sugiere la implantación de:

- Filtros conformadores de tráfico que modifican el patrón de tráfico suavizando las ráfagas del tráfico no HPC (con y sin QoS).
- Reducción del tamaño del paquete enviado para tráfico no HPC (el peor caso ocurre con múltiplos del tamaño de la mtu) en caso de ser posible.

7 Piloto

En el siguiente capítulo se despliega un piloto experimental con el objetivo de comprender y estudiar las diferentes problemáticas que se han analizado en capítulos anteriores.

7.1 Propuesta de análisis

A partir del estudio expuesto en anteriores capítulos sobre las problemáticas a encontrar ante la ejecución de trabajos HPC en entornos Cloud multipropósito, se propone un piloto compuesto por un par de servidores conectados por un conmutador, en los que a partir de la virtualización de servidores, se harán pruebas de diferente naturaleza entre las MV de ambos lados de la infraestructura.

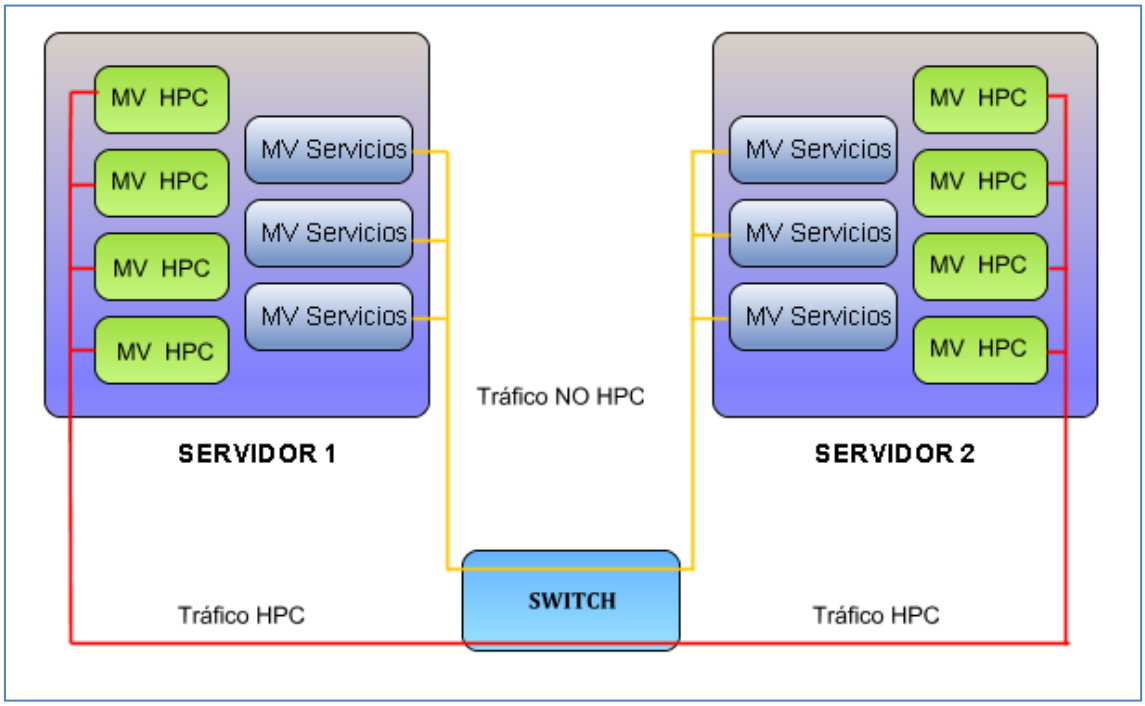

Ilustración 24 - Esquema piloto propuesto

Por simplicidad se ha escogido un tipo de benchmark sintético que simule el comportamiento de los tipos de trabajos HPC que podamos ejecutar y que obtenga los valores de las métricas que deseamos a analizar (retardos, anchos de banda y variaciones). No todas las aplicaciones de alto rendimiento tienen las mismas necesidades. Hay que tener en cuenta que muchas tendrán requisitos de computación y comunicación diferentes, algunas pueden ser más comunicativos y otras menos comunicativos. El paso de mensajes que realicen estos entornos paralelos puede ser más grande o más pequeño. Por ello es conveniente tanto analizar el ancho de banda como los retardos existentes en la infraestructura de red desplegada. Y no solo eso, también es necesario tener en cuenta la variación de retardo (*jitter*) con el objetivo de que sea mínima, puesto que los trabajos de HPC requieren que las diferentes latencias de los nodos sean homogéneas.

En primer lugar se estudiará el estado de la arquitectura de red sin interferencias, es decir ejecutando exclusivamente programas HPC, y se comprobará el impacto y *overhead* de la virtualización en las comunicaciones y el rendimiento del protocolo TCP en estas.

En segundo lugar se estudiará el comportamiento de la infraestructura cuando se realiza una tarea distribuida de HPC en convivencia con otras de distinta finalidad que provoquen tráfico en la infraestructura (pruebas con interferencias). Con el fin de mitigar la degradación que las interferencias provoquen al tráfico de altas prestaciones, se propondrá:

- Implementar técnicas de QoS en el extremo de ambos servidores para priorizar tráficos y conseguir heterogeneidad en la red de computación.
- Aplicación de optimizaciones también de calidad de servicio en la zona del conmutador.

Con la finalización del presente piloto se pretende analizar el efecto de determinadas medidas seleccionadas de entre las propuestas incluidas en el capítulo anterior con el fin de mejorar las condiciones de funcionamiento de los tráficos HPC en infraestructuras *cloud* multipropósito.

7.2 Diseño y metodología del experimento

Los principales puntos críticos de la red que o bien entran en conflicto con otros tráficos, o bien retardan las comunicaciones de programas de alto rendimiento serán las siguientes:

- *Los protocolos de la capa de transporte y de la capa de red.* Problemas relacionados con el rendimiento de pila TCP/IP y sus algoritmos de congestión para tráficos elevados. Existen recomendaciones para realizar ajustes de parámetros la pila TCP (aunque la mayor parte de recomendaciones están orientadas a incrementar la latencia), y múltiples algoritmos de congestión que pueden funcionar mejor en contextos con RTT muy pequeño.
- *Las tarjetas de red y los servidores.* Debido a que existirá contención en la entrada/salida de las NIC entre paquetes de los diferentes tráficos que hayan sido generados por un conjunto heterogéneo de máquinas virtuales, el rendimiento de los trabajos HPC puede verse comprometido. Existen soluciones para priorizar estos trabajos en las colas de entrada y salida de los servidores físicos (dom0), como por ejemplo las aplicaciones de QoS para Linux.
- *Los conmutadores.* Aquí probable congestión de los búferes de los conmutadores será podrá generar contención de paquetes llegando incluso al descarte. Por lo tanto, es también necesario establecer aquí unas disciplinas de QoS análogas a la parte del servidor. Adicionalmente, hay que tener en cuenta que los conmutadores introducen irremediablemente cierta latencia a las comunicaciones, por lo que a mayor número de conmutadores atraviese un determinado tráfico, mayor latencia existirá. Valores típicos de latencias para switches 1 Gigabit Ethernet es de **26,24 microsegundos** para un tamaño de frame de 1518 bytes "*Juniper switch aces initial test*" ([21], 2008).

Se ha de tener en cuenta el *overhead* de la virtualización en las diferentes máquinas, si bien no se experimentará en profundidad ninguna optimización sobre este tema. Si bien el acceso en modo privilegiado de los *domU* a la tarjeta de red aumentaría mucho el rendimiento de las comunicaciones de ésta; dichas técnicas no son del todo seguras, y podrían hacer que la transmisión de datos fuese inestable.

La metodología de los experimentos es sencilla. Mediante la utilización de un *benchmark* se medirá los distintos parámetros de red en escenarios donde no exista competencia

de tráfico. Tras esto, se procederá a hacer las mismas mediciones, aplicando interferencias a la ejecución del *benchmark* mediante generadores de tráfico desde máquinas virtuales vecinas. Se aplicarán optimizaciones de QoS a este escenario, y se compararán resultados.

El *benchmark* escogido para medir las diferentes latencias entre los nodos ha sido ***Intel IMB MPI***. Este es una suite de test sintéticos implementada por Intel, y que es ampliamente utilizada en la literatura afín a este proyecto para analizar como escalan las latencias y el ancho de banda entre las diferentes tipos de mensaje de los nodos. Es posible tanto analizar comunicaciones punto a punto; como los diferentes tipos de funciones colectivas ampliamente utilizadas en MPI (*broadcast* o *AlltoAll*).

Se utilizarán los test PingPong y PingPing, pertenecientes al primer grupo (comunicaciones punto a punto), para medir retardo unidireccional y latencia entre los diferentes nodos desplegados.

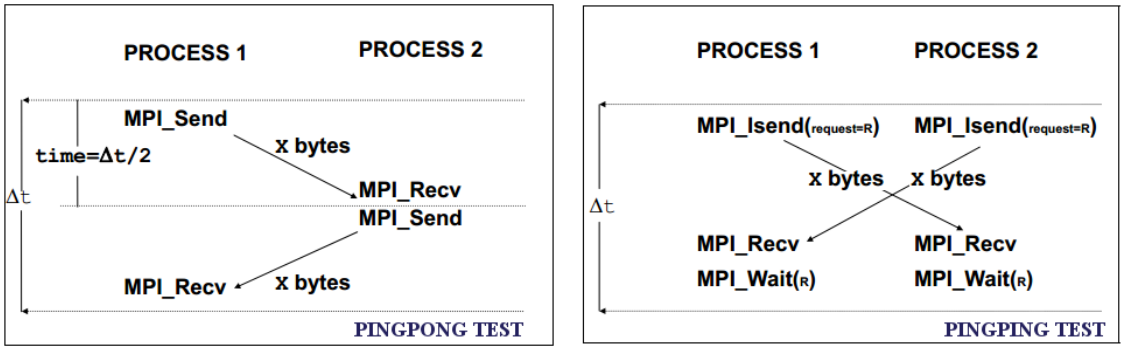

Ilustración 25 - Test de Intel IMB MPI PingPong y PingPing

7.3 Material, despliegue y configuración

El piloto contendrá los siguientes dispositivos hardware:

- 2 servidores Intel Xeon E5520 a 2.27 GHz con cuatro cores, hyperthreading y 16GB de RAM y NIC Intel Corporation 82576 Gigabit Network Connection
- 1 servidor adicional para la generación de tráficos *iperf*
- 1 conmutador Juniper EX 4200.
- Interconexión mediante red 1 *gigabit ethernet*.

En los servidores se ha instalado el sistema operativo Centos 6.3 en cada nodo; sobre lo que se ha instalado la tecnología de virtualización Xen 4 (en su versión paravirtual). Se han creado máquinas virtuales en cada servidor destinadas a la computación HPC, siendo una de ellas el *front-end*, y desplegando para mayor comodidad en la ejecución de trabajos MPI un servidor NFS a partir del cual compartan los ficheros de sus trabajos distribuidos. Adicionalmente, se ha instalado la implementación de MPI openMPI versión 1.6; junto con los *benchmark* Intel IMB MPI.

A la vez, se han creado una serie de máquinas destinadas a la ejecución de otros servicios (serán las inferirán interferencias en las comunicaciones) en las que como software de generación de tráfico se ha optado por *iperf* (genera tráfico UDP y TCP para medir el ancho de banda de una red) procediéndose a su instalación.

Se ha configurado la red de todas estas mediante interfaces tipo bridge, agrupando las máquinas en dos VLANs según su propósito: una dedicada al tráfico HPC y otra dedicada a servicios. Esto se ha realizado para garantizar la independencia de los diferentes tipos de MV desplegados en el piloto.

Para la implementación de QoS en los servidores se ha utilizado el mecanismo de tc (traffic control) proporcionado por Linux por defecto.

Test 0: Medida de parámetros de red sin contienda en escenarios con y sin virtualización

En estas medidas preliminares se pretende cuantificar los valores de los parámetros en el punto de partida (sin tráfico de competencia) así como la penalización que nos imponen los sistemas de virtualización:

Una vez desplegado y configurado el piloto, se ha realizado una primera prueba con el benchmark *Intel IMB MPI*, con el objetivo de conocer el estado de la latencia y ancho de banda de las comunicaciones punto a punto entre los dom0 de cada servidor, en los cuales como se sabe su rendimiento debería estar muy próximo al de una máquina no virtualizada, y compararlo así con las medidas que se obtienen en las comunicaciones entre *domU* alojados en diferente máquinas físicas. Dicha prueba ha sido realizada sin la generación de tráfico interferente no HPC (puede haber tráfico residual proveniente de NFS o de la conexión por SSH con la que se accede a la máquina).

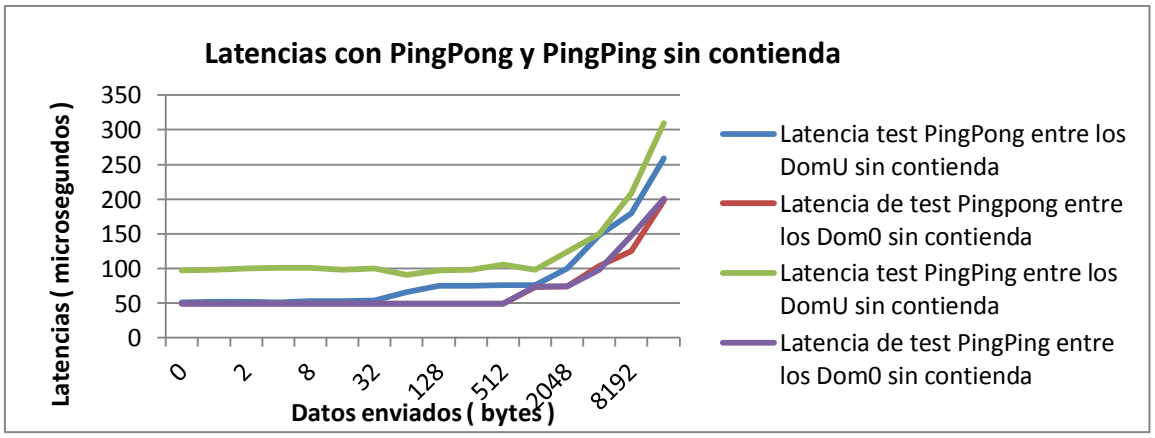

Ilustración 26 – Latencia inicial de los domU y dom0 sin contienda

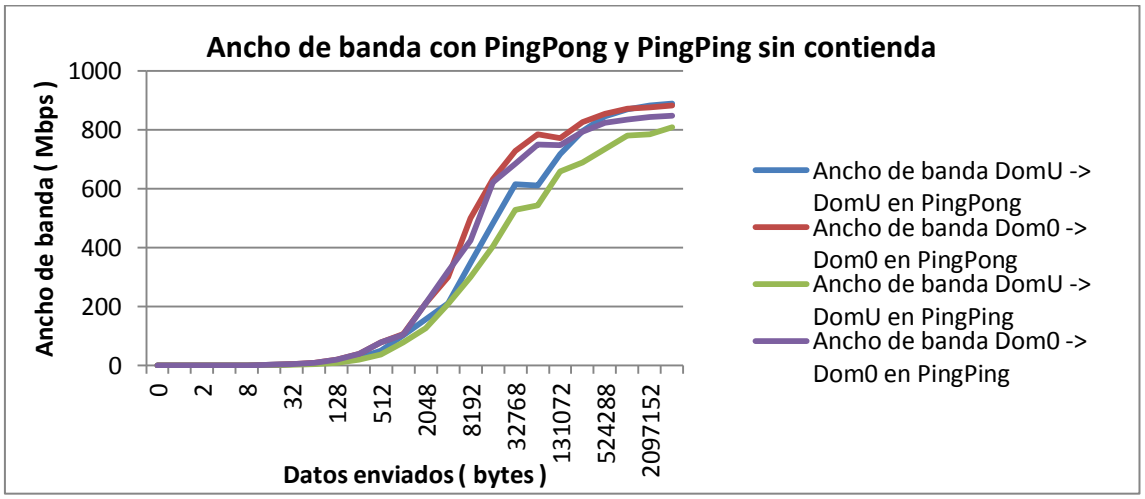

Ilustración 27 – Throughput inicial de los domU y dom0 sin contienda

Como se puede ver en las anteriores imágenes, las latencias mínimas del *dom0* para mensajes pequeños del son de aproximadamente 50 microsegundos. Este tiempo vendrá dado principalmente por el procesado de los paquetes TCP/IP en la pila en cada servidor (los mensaje

MPI irán bajo TCP/IP) y por el retardo añadido por el conmutador (aproximadamente 20 microsegundos en este caso).

A partir de tamaños de paquete superiores a 1024 bytes se aprecia un incremento de la latencia exponencial derivada del overhead derivado de la transmisión del propio paquete (un paquete de 2048 bytes tardaría aprox. 65 ms en transmitirse a 1Gbps).

En cuanto al ancho de banda escala a valores cercanos a 1Gbps observándose el efecto de los mecanismos de control de congestión cuando nos acercamos a los límites de la capacidad de la línea.

En el caso del *domU*, se observan valores de latencia mayores derivados del overhead que introduce la capa de virtualización, más visible cuando se realiza comunicación bidireccional (test PingPing). En cuanto ancho de banda, también se observa una disminución apreciable.

Test 1: Medida de parámetros de red con contienda de flujos TCP y UDP

Partiendo de este escenario, se han realizado diferentes medidas de los test *Intel IMB* entre las máquinas virtuales para HPC, a la vez que se han realizado diferentes llenados de caudal UDP entre las máquinas de servicios (alojados a su vez en diferentes servidores físicos) mediante la herramienta *iperf*. El objetivo ha sido observar la degradación que se presentará en las comunicaciones ante la compartición de recursos de red y sistemas por ambos tráficos.

Como se puede ver en la siguiente figura, la degradación que provoca una pequeña interferencia UDP bidireccional de 200Mbps sobre el tráfico MPI es bastante elevada. Y no solo eso, si se repite un mismo test un determinado número de veces consecutivas, la degradación puede ser aún mucho mayor, haciendo que las latencias en el envío de datos pequeños sea variable, y entre ejecución y ejecución no se consiga el mismo patrón de datos.

Esta variación de tráfico entre las diversas iteraciones está vinculada a los efectos que producen los mecanismos de *coalescing* ya que desapareció en las pruebas posteriores mediante la deshabilitación del mismo.

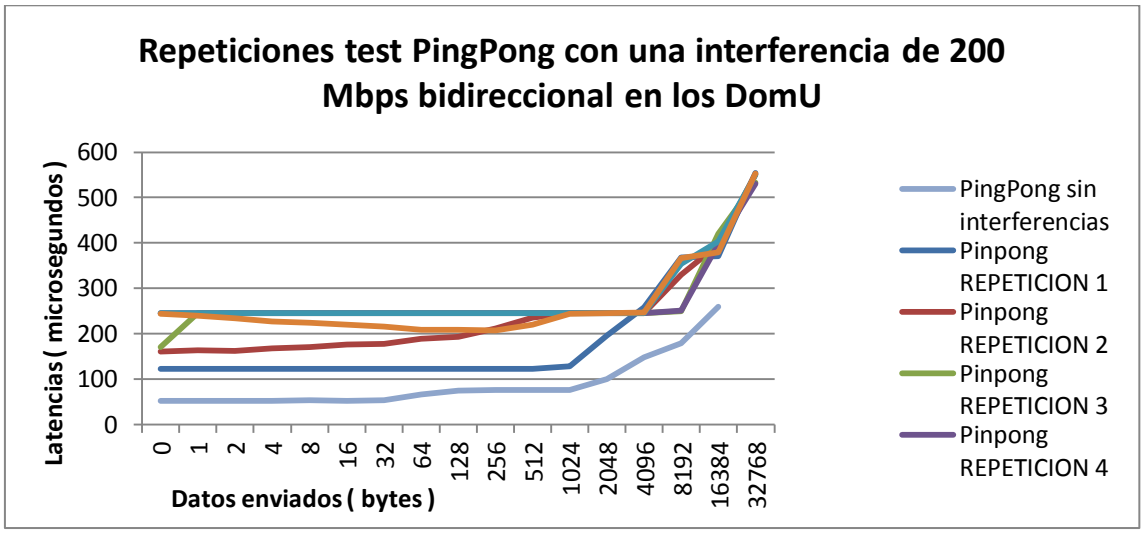

Ilustración 28 - Problemas iniciales entre el tráfico MPI y una pequeña interferencia UDP de 200Mbps: variación de resultados

Si se aumenta el caudal de esta interferencia UDP, los resultados a los test mostrarán aún más degradación en las pequeñas latencias.

En caso de que las interferencias sean de tipo TCP, la afectación al tráfico HPC es mucho mayor dado que la herramienta no permite generar un flujo de un ancho de banda concreto, sino que TCP tratará de maximizar el ancho de banda utilizado debido a sus algoritmos de congestión, generando una interferencia elevada al tráfico HPC. Por contraposición el tráfico generado mediante UDP tiene un *bitrate* especificado en la propia aplicación de *iperf*.

A partir de este fenómeno, el objetivo siguiente será analizar y mitigar el impacto de estas interferencias en el tráfico prioritario HPC para acercar lo máximo posible los valores de latencia y ancho de banda a los experimentados por los *domU* del escenario inicial.

Test 2: Modificación de parámetros: coalescing, checksm, tcp_low_latency

Como se indicó en las conclusiones del apartado anterior se determinó que deshabilitando la opción "*interrup coalescing"* en las tarjetas de red de los dom0 se obtenía un resultado determinista en las medidas de latencia y ancho de banda. Una explicación posible a esto es que dicha opción, que acelera las tasa de envío de paquetes por parte de la NIC, puede provocar ráfagas de tráfico, cuyos picos congestionan o bien los búferes de salida de la tarjeta de red, o bien las colas de salida del conmutador.

Como se puede ver en la figura, si se repite el experimento un cierto número de veces, las latencias resultantes presentan un comportamiento determinista.

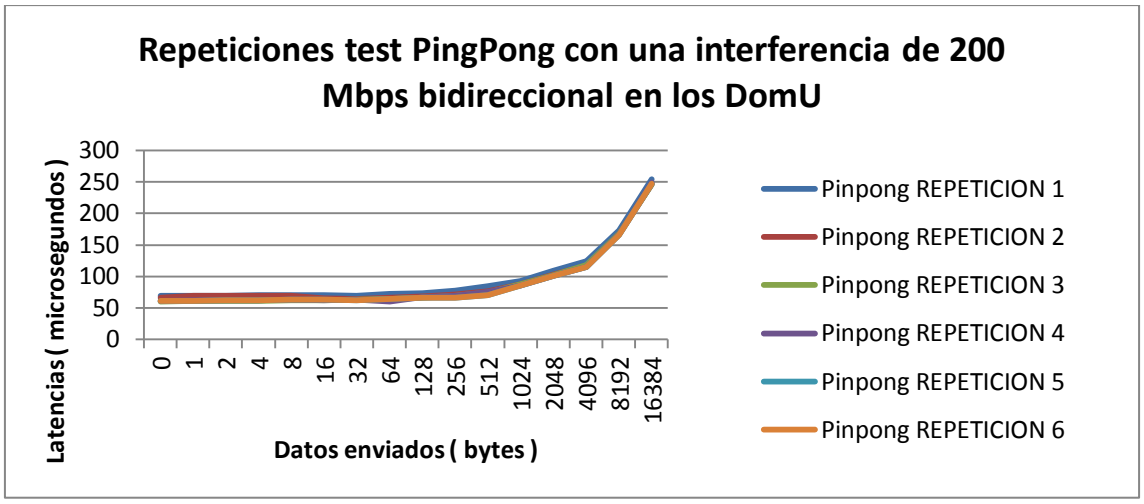

Ilustración 29 - Pingpong con interrupciones UDP de 200 Mbps, tras deshabilitar coalescing

Adicionalmente se han añadido las siguientes mejoras en las comunicaciones:

- Se ha habilitado en el kernel de los *domU* y *dom0* la opción *tcp_low_latency*. Esto hará que los paquetes se salten las colas TCP intermedias de entradas existentes en el servidor, y vayan directamente a la cola de recepción.
- Se ha deshabilitado el *checksum* en los *domU*, para que la suma de verificación no se realice dos veces (ya se realiza en el dom0).
- Se utiliza el algoritmo de congestión CUBIC, que funciona bien en tiempos de RTT pequeños como es el caso que nos ocupa.

A continuación se procedió a la repetición de los *benchmarks* y se observa que la degradación en los test *Intel IMB* disminuye pero persiste, apreciable sobre todo en los retardos del envío de datos pequeños.

En el caso del ancho de banda, la aplicación MPI escalará bien hasta ocupar todo el caudal que esté disponible, punto donde se estabilizará.

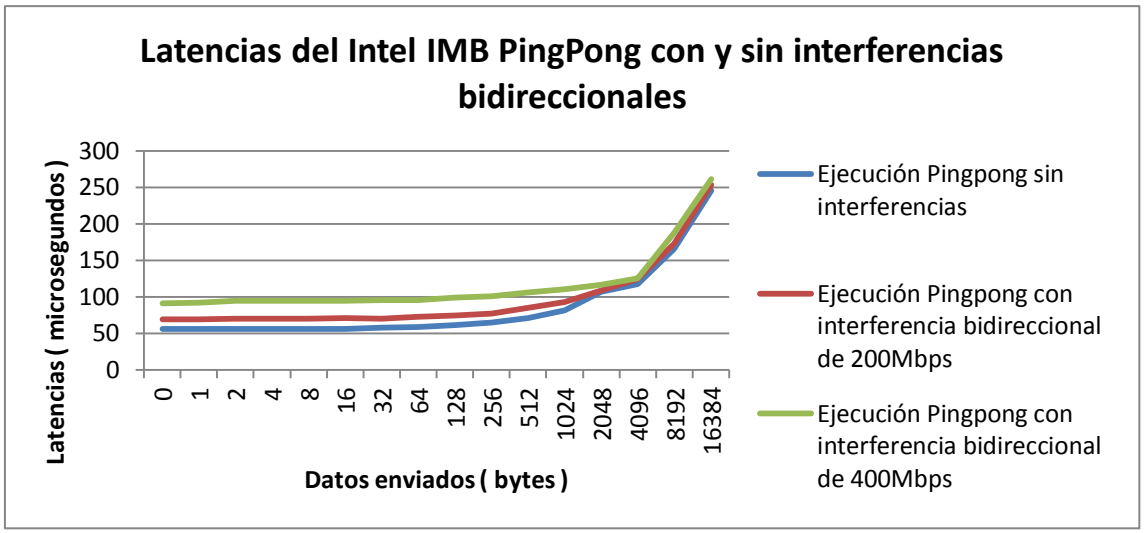

Ilustración 30 - Latencia ante las optimizaciones iniciales

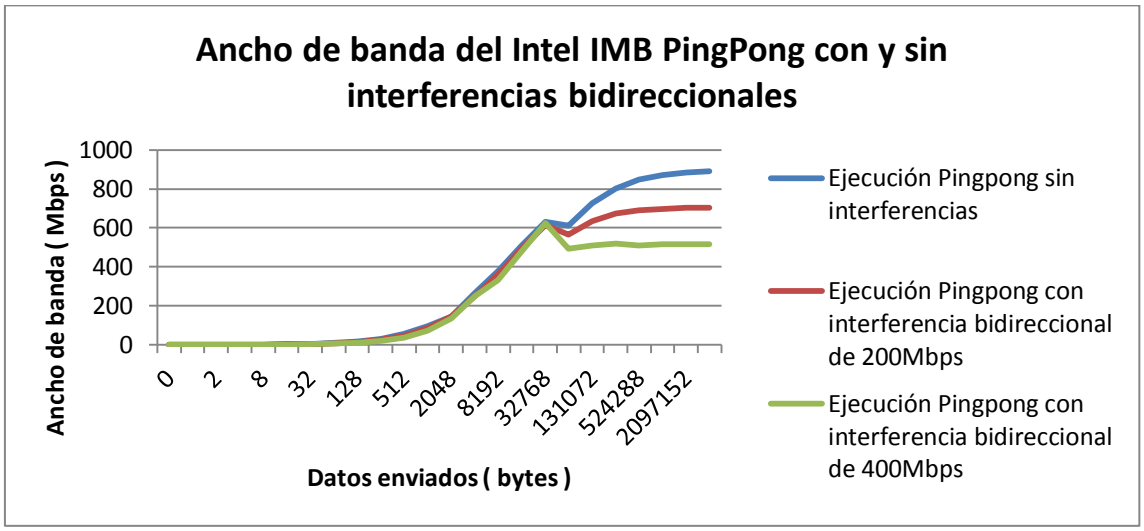

Ilustración 31 - Ancho de banda ante las optimizaciones iniciales

Test 3: Aplicación de mecanismos de calidad de servicio en el conmutador

Bajo estas circunstancias se propone la utilización de mecanismos de calidad de servicio en el conmutador. Se habilita la distinción en el tratamiento del tráfico según de su red IP de origen, de HPC o de servicios, introduciendo los paquetes en una cola de salida con mayor o menor prioridad según este origen. Se utiliza el planificador *Strict-Priority (SP)* para despachar las colas según su prioridad estática, es decir, mientras existan paquetes en la cola más

prioritaria, solo se enviarán por el puerto de salida dichos paquetes. Como algoritmo de congestión se utilizará *Weighted tail drop (WTD).*

Como resultado, no se observan mejoras substanciales en las comunicaciones con interferencias, lo que parece indicar que la competencia se produce no en los búferes del conmutador, sino en los propios servidores. Para asegurar que los mecanismos de QoS funcionan correctamente, se propone un nuevo experimento: para evitar la ofuscación de mejoras en los problemas de contención que posiblemente existan en la NIC, se realiza una interferencia desde otro servidor físico diferente hacia uno de los servidores donde se ejecute el trabajo MPI. De tal manera el tráfico no competirá en los interfaces de salida de los servidores Linux, sino en los búferes del puerto de salida del conmutador. Se verificó la transmisión con flujos de 1Gbps con el fin de forzar la existencia de contención y descarte en dicho punto.

Como se observa en las siguientes figuras, los mecanismos de calidad de servicio son realmente palpables cuando existe congestión en la red. Ante una interferencia de 1Gbps desde el nuevo servidor externo, las técnicas de QoS en el conmutador permiten que el tráfico se comporte como ante una interferencia de 500Gbps. Adicionalmente, las latencias ante esta interferencia no es muy diferente al comportamiento sin tráfico en competencia, por lo que se puede extraer que el siguiente mayor problema de contención a resolver residirá en la NIC.

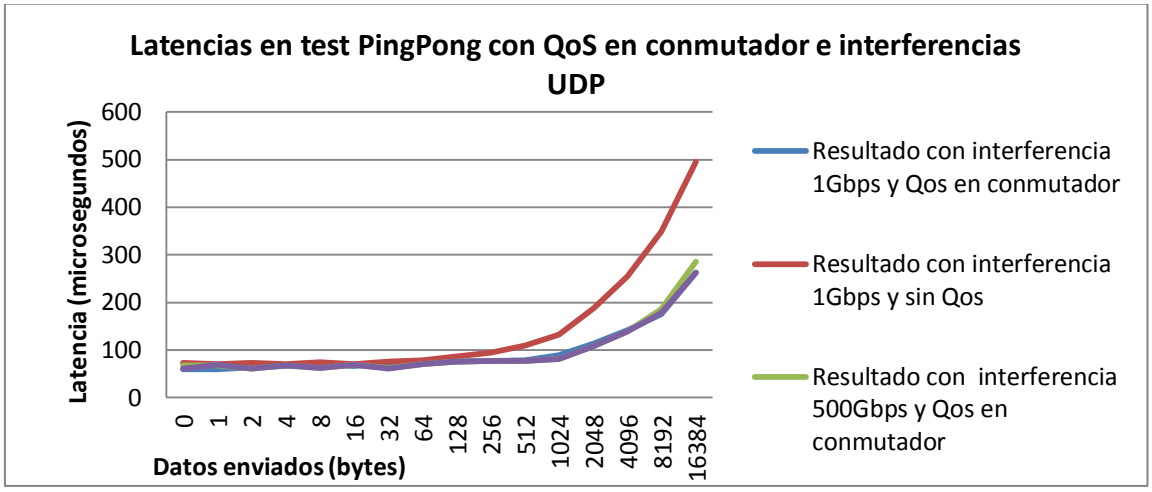

Ilustración 32 - Latencias para PingPong con QoS en conmutador

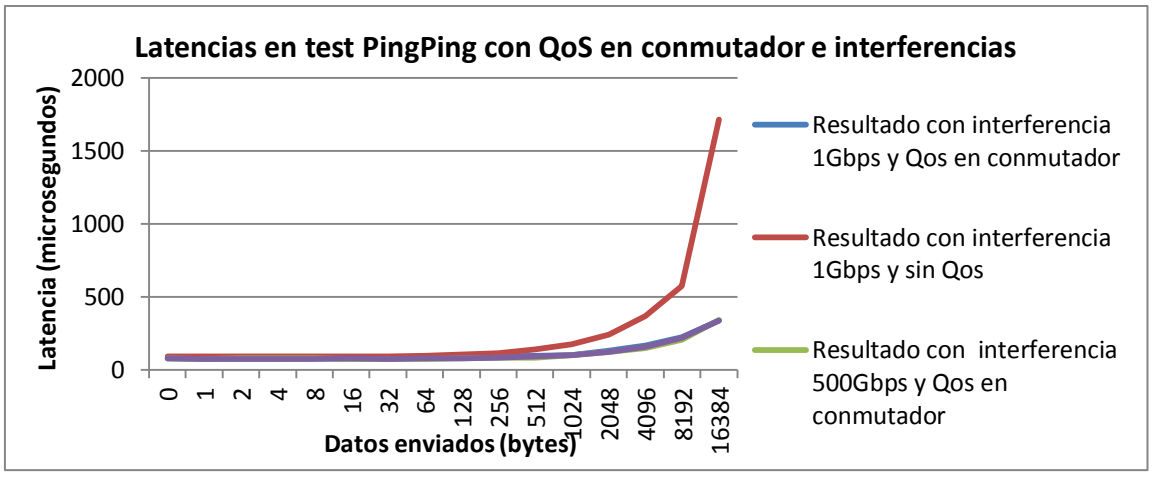

Ilustración 33 - Latencias para PingPing con QoS en conmutador

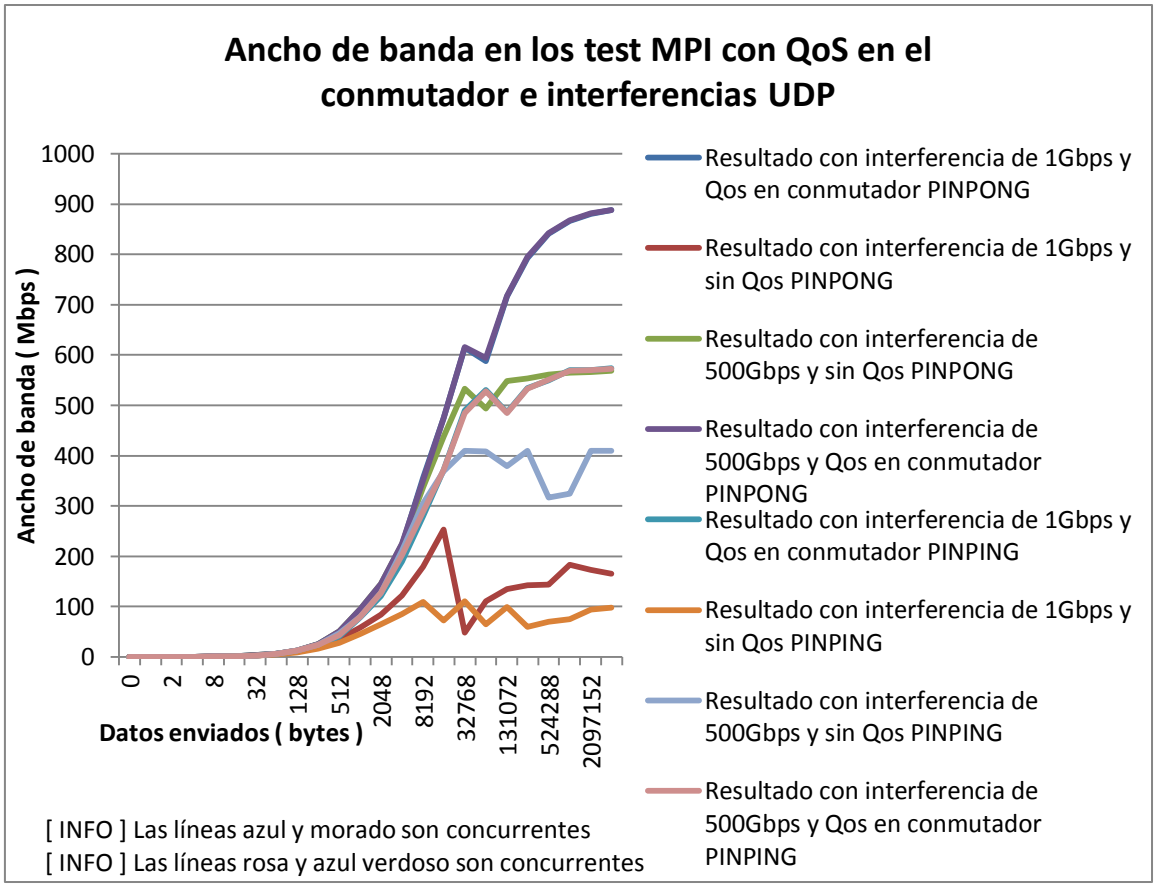

Ilustración 34 - Latencias para PingPing y PingPong con QoS en conmutador

Test 4: Aplicación de mecanismos de calidad de servicio en los servidores

A continuación se aplicarán mecanismos de calidad de servicio en la tarjeta de red con el objetivo de combatir la contención detectada en ella. Para estas pruebas volvemos al escenario habitual (comunicaciones entre las dos máquinas físicas que tienen las diferentes MVs para servicios y para HPC).

Mediante la herramienta *"tc"* de Linux, se configuran tres colas priorizadas de salida en el *dom0*, una para el tráfico proveniente de las máquinas de HPC, otra para las máquinas de servicio y una tercera para el resto del tráfico. En cada una de ellas, la estrategia de planificación escogida ha sido *pfifo_fast*. Tal y como ocurren en los conmutadores, también en este caso los mecanismos de calidad de servicio apenas son apreciables cuando no existe congestión. Por ello, en la siguiente prueba la interferencia UDP trataremos de congestionar la red mediante dos procesos en cada máquina de servicio que reservarán un caudal de 500Mbps cada uno. Adicionalmente, se ha variado el tamaño del datagrama para observar el comportamiento en las diferentes pruebas.

Como se puede ver en las siguientes gráficas, los mecanismos de QoS si funcionan reduciendo considerablemente las latencias, y aumentando el ancho de banda. La mejora es significativa, pero quizás no sea lo suficiente. ¿Cuál es el motivo de que aun aplicando QoS en la NIC y en el conmutador existan valores tan altos de retardo?

A medida que el tamaño de datagrama es menor, los retardos apreciados son menores. Los planificadores existentes en las colas de salida de la NIC y del conmutador harán bien su trabajo: despacharán el paquete siempre y cuando haya alguno esperando en la cola prioritaria. El problema es que a diferencia del programa MPI (PingPong y PingPing) que estamos probando, en la que la comunicación se interrumpe continuamente o bien para iniciar una nueva iteración de envío de datos o bien para cambiar de tamaño; las comunicaciones interferencia UDP realizadas con el *iperf*, están continuamente emitiendo grandes ráfagas de datos. Por ello, existirá una gran probabilidad a que cuando un paquete HPC llegue a una cola vacía, este tenga que esperar a que la tarjeta de red finalice la gestión de un determinado paquete de servicios. Obviamente, este efecto de reducción de la latencia será más visible cuanto menor sean los tamaños de los paquetes a transmitir del tipo de tráfico no HPC; puesto que ante grandes flujos de trabajo MPI, más probabilidades existen de que la cola no esté vacía, y que continuamente se estén despachando paquetes HPC.

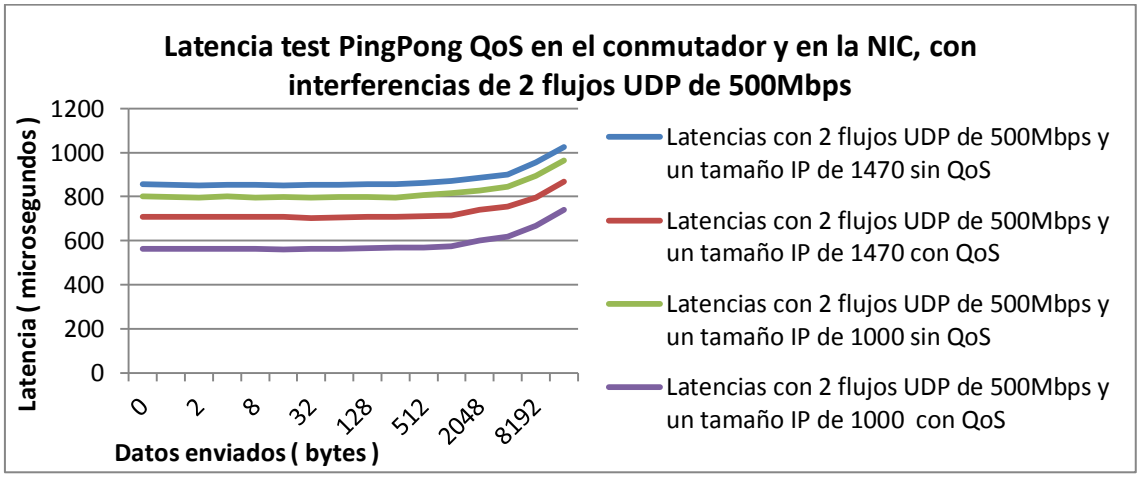

Ilustración 35 - Latencias para PingPong con QoS en NIC y conmutador, bajo congestión por UDP.

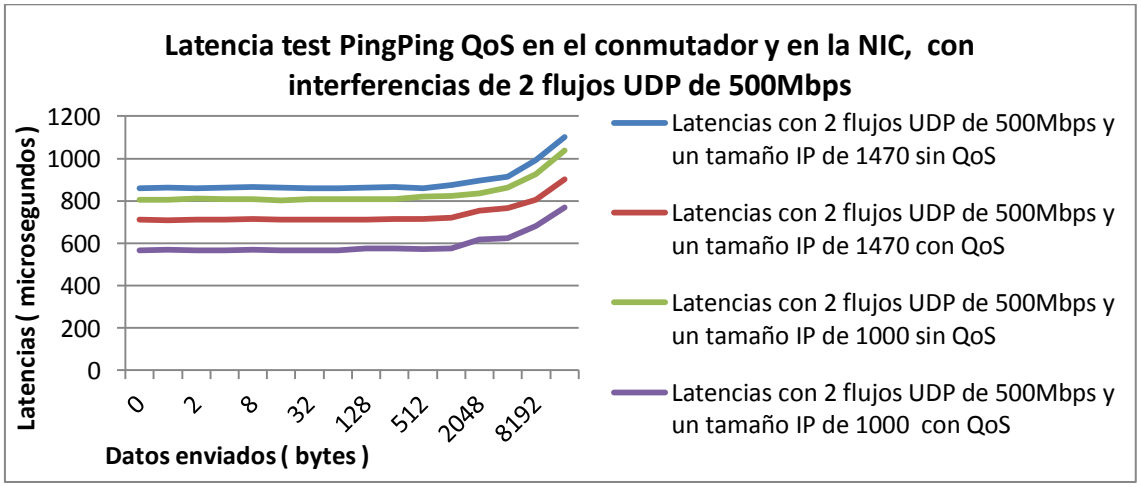

Ilustración 36 - Latencias para PingPing con QoS en NIC y conmutador, bajo congestión por UDP.

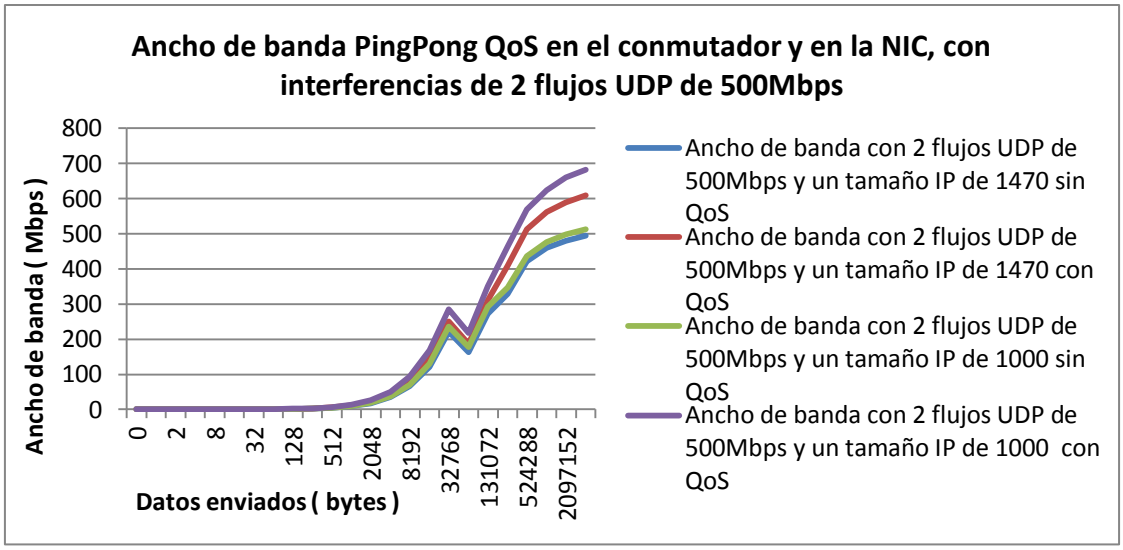

Ilustración 37 - Ancho de banda para PingPong con QoS en NIC y conmutador, bajo congestión por UDP.

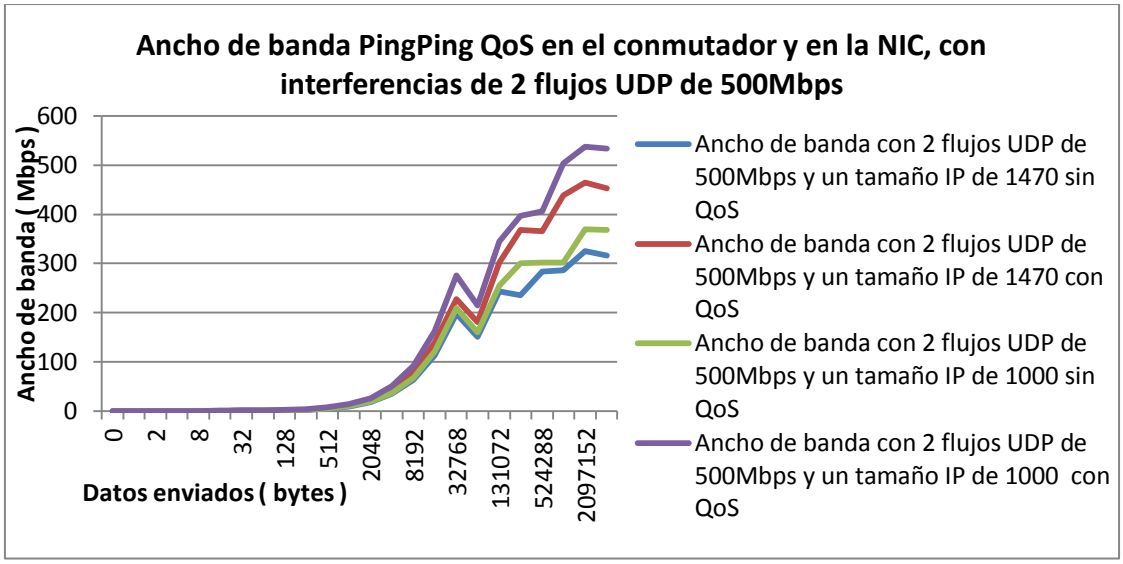

Ilustración 38 - Ancho de banda para PingPing con QoS en NIC y conmutador, bajo congestión por UDP.

También se han realizado pruebas con otros mecanismos de planificación de cola más allá de las pfifo_fast, como con la *stochastic fair queuing (SFQ)*. Debido a que éstas (distribución equitativa de paquetes entre flujos diferentes) aprovecharán más el tamaño de los paquetes en el tráfico de las MV de servicio, los resultados de aplicar SFQ en las máquinas de servicio serán ligeramente peores que con *pfifo_fast*. Esta disciplina en cambio, si puede ser interesante para gestionar los tráficos de las máquinas virtuales HPC, debido a que podría favorecer la convivencia de varios flujos distintos de HPC en el mismo servidor físico.

Test 5: Generación de contención con flujos TCP

Hasta ahora, se han realizado interferencias unidireccionales con UDP de un caudal determinado. A continuación se realizarán las mismas pruebas que en el caso anterior, pero esta vez con interferencias mediante el protocolo TCP, teniendo en cuenta que cada flujo tratará de

ocupar al máximo el canal; debe tenerse en cuenta que el servidor deberá gestionar la recepciones de ACKs y los mecanismos de congestión de este.

Adicionalmente se desactivara el mecanismo *TSO (TCP segmentation offload)* de los *dom0*, dejando solo activado el mecanismo de segmentación de paquetes IP *GSO (generic segmentation offload)*. Esto hará que las latencias sin mecanismos de QoS sean aún peores, debido a que se disminuirá la aceleración resultante, pero mejorará los tiempos en el caso de aplicar estos mecanismos. Esto es debido a que si se practica segmentación a nivel de IP; y también a nivel TCP, se supone que existirán un mayor número de paquetes a competir en la salida de la NIC (y mayores ráfagas); por lo que se puede retrasar paquetes prioritarios tal y como se ha comentado anteriormente. Pero la no presencia de mecanismo alguno de segmentación hace que la calidad de las comunicaciones se degrade, por lo que es preciso mantener la segmentación IP GSO activada.

En este caso, también se mejora al disminuir el tamaño de paquete (*MTU*). Como se puede ver en las siguientes imágenes, para tráficos TCP los mecanismos de calidad de servicio suponen una mejora drástica tanto en las latencias, como en el escalado de los anchos de banda.

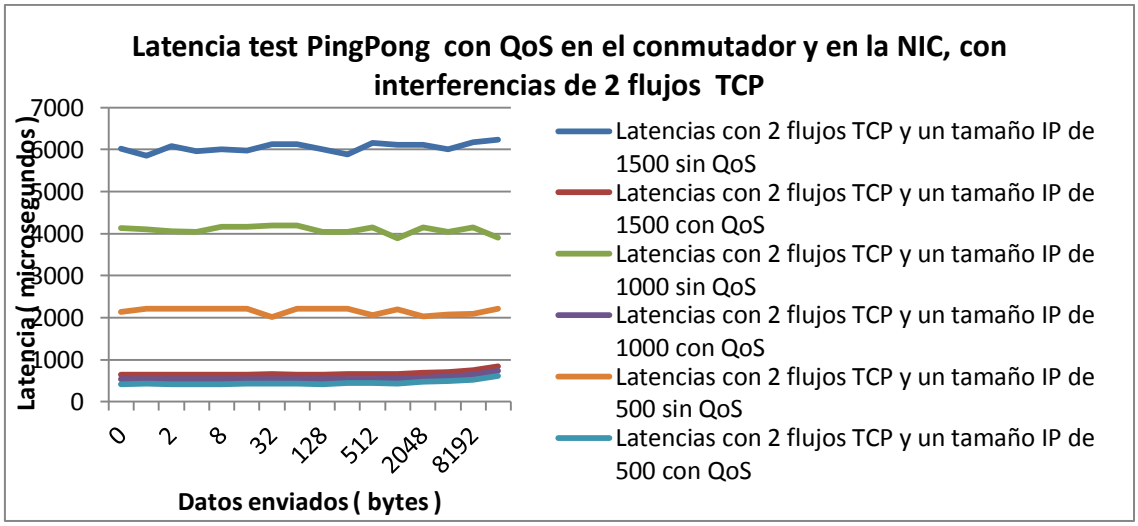

Ilustración 39 - Latencia para PingPong con QoS en NIC y conmutador, bajo congestión por TCP.

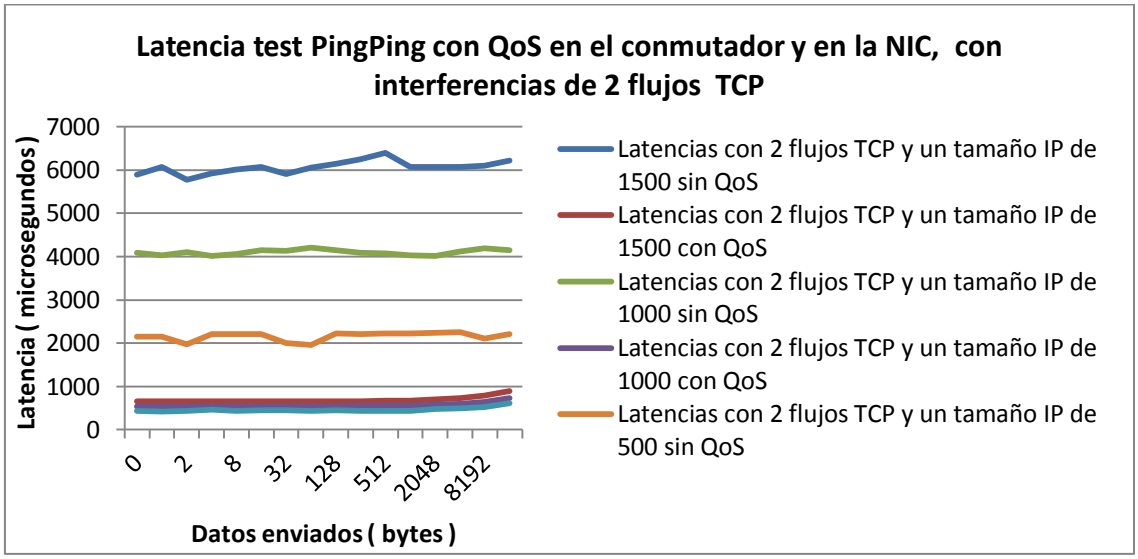

Ilustración 40 - Latencia para PingPing con QoS en NIC y conmutador, bajo congestión por TCP.

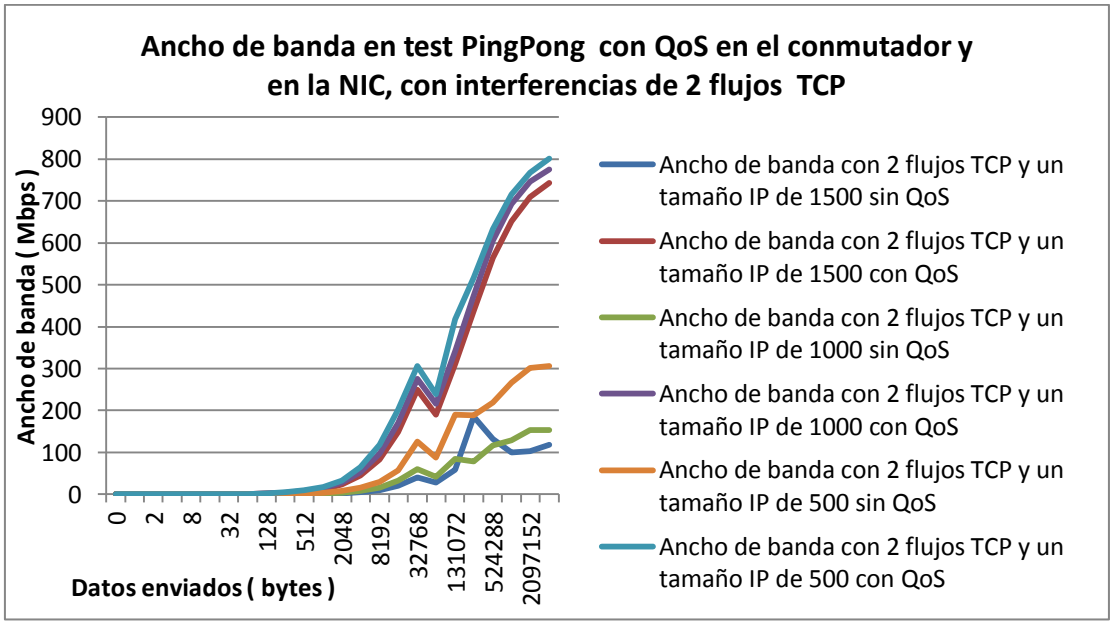

Ilustración 41 - Ancho de banda para PingPing con QoS en NIC y en conmutador, bajo congestión por TCP.

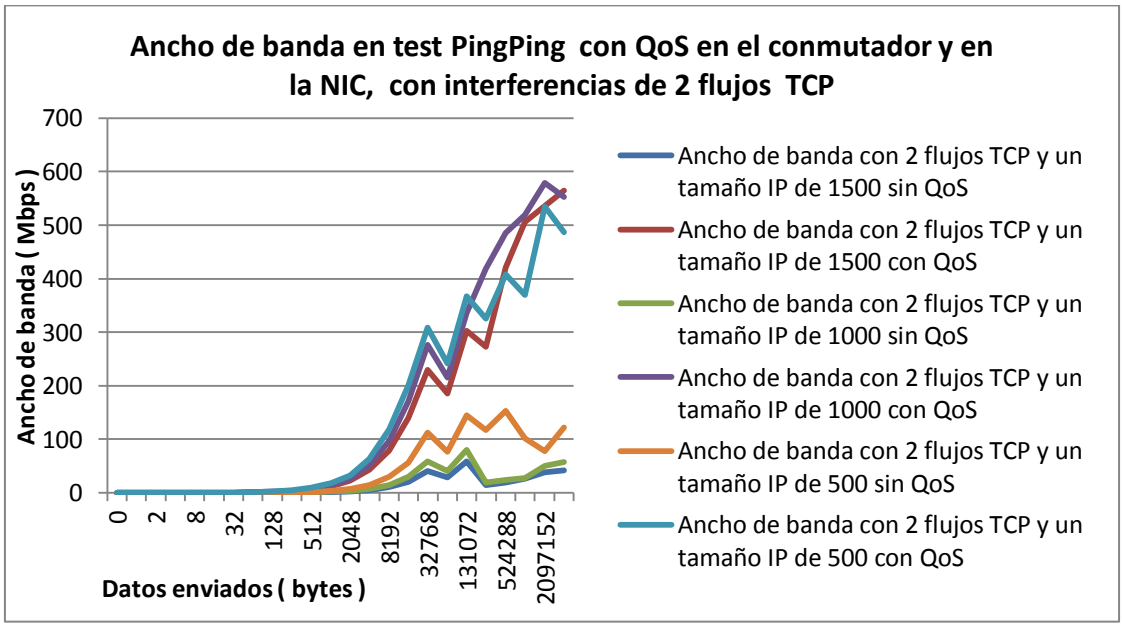

Ilustración 42 - Ancho de banda para PingPing con QoS en NIC y conmutador, bajo congestión por TCP.

De estos resultados se extrae que se necesita un mecanismo de espaciado de tráfico para las comunicaciones de servicio, es decir que se necesita reducir la tasa de emisión de estos flujos, para disminuir la posibilidad de que cuando llegue a la NIC o a una cola vacía de salida de un conmutador un paquete prioritario (HPC), este tenga que esperar a que el dispositivo en cuestión termine de gestionar el envío de un paquete no prioritario.

El objetivo de un "*traffic pacer*" es la de proporcionar una tasa de bits constante, retrasando el tráfico de ciertos paquetes. Como este no será capaz de conocer los tiempos de llegada de los futuros paquetes, no es perfectamente capaz de determinar los tiempos de transmisión para proporcionar una tasa de bits constante. En lugar de esto, se utiliza un proceso adaptativo para determinar en qué medida los paquetes pueden ser retrasados.

No existen muchas herramientas software para implementar esto en las "*qdisc*" implementadas en Linux; y el existente es para versiones muy específicas de sistema operativo y de librería *iproute* (es el caso de *pspacer*). Además, en la literatura normalmente implementan "*pacing*" en el tráfico a través de dispositivos hardware específicos (como es el caso de la "*NetFPGA*"), se supone que debido a que realizarlo mediante software puede provocar la inclusión de aún mayores latencias.

Otra solución sería reducir el ancho de banda disponible para las colas de servicio mediante la disciplina de cola *TBF* (*Token Bucket Filter*). Esto reservaría cierto margen de maniobra de cara a ejecutar tráfico HPC; pero limitaría el uso de la infraestructura cuando este no se esté ejecutando.

Como se ha comprobado, uno de los puntos críticos de las infraestructuras *cloud* multipropósito en los que se quiere introducir la ejecución de trabajos HPC, son las tarjetas de red compartidas. Quizás una buena opción sería añadir a la infraestructura una NIC adicional, y así poder disgregar el tráfico de HPC exclusivamente en una de ella y el tráfico de las máquinas de servicio y gestión por la alternativa.

7.4 Análisis de los resultados del piloto

Del análisis de los diferentes experimentos llevados a cabo pueden extraerse diferentes conclusiones que deben ser objeto de una análisis en mayor profundidad a posteriori. Enumeramos a continuación las más relevantes:

- Los retardos obtenidos en condiciones habituales en una red *cloud* de propósito general son excesivamente elevados para la ejecución de trabajos HPC aún en el caso de uso en exclusividad con el propósito de ejecución de HPC.
- La capa de virtualización introduce *overhead* adicional no despreciable.
- Los patrones y protocolos del tráfico de competencia son relevantes para evaluar su influencia sobre el comportamiento del tráfico HPC.
- La aplicación de mecanismos convencionales de QoS aporta ventajas para el tráfico sensible a latencia.
- Los diferentes mecanismos y protocolos involucrados en la conmutación de tramas *ethernet* utilizados tradicionalmente han estado vinculados habitualmente a maximizar el *throughput* penalizando en ocasiones parámetros como la latencia.
- Existen pequeñas mejoras adicionales que pueden ayudarnos a mejorar en cierta medida las condiciones del tráfico sensible a latencia como se ha visto: *tcp_low_latency*, *coalescing* o los mecanismos de *offloading* de las NICs; aunque deberá ser analizado con cautela su posible perjuicio en otros ámbitos (sobrecarga de CPU en flujos muy comunicativos).
- El control de los patrones de tráfico de competencia debe ser analizado, pudiendo ser recomendable la incorporación de mecanismos de *pacing*.
- Unos de los puntos que se ha presentado como más crítico en la convivencia de diferentes tipos de tráfico es la NIC del dispositivo.

8 Conclusiones y líneas futuras

8.1 Conclusiones

El despliegue de aplicaciones sensibles a latencia, como en el caso de los procesos de HPC en entornos *cloud* multipropósito, recomienda la revisión de los diversos elementos de red atravesados por su tráfico.

Del presente estudio hemos podido verificar que es necesario una revisión de todos los elementos implicados en las comunicaciones: dispositivos físicos, protocolos y aplicaciones software.

A lo largo del mismo se ha realizado un estudio exhaustivo de las tecnologías de red implicadas, con el fin de evaluar los posibles puntos de contención o descarte de paquetes.

Se ha realizado un análisis pormenorizado del estado del arte en los ámbitos de ultra baja latencia en las comunicaciones en LAN, virtualización y calidad de servicio, complementando la visión existente en relación a temáticas como la compartición de usos en la red y los servicios, la gestión de los búferes en los diversos componentes de la red, la utilización de protocolos de transporte más adecuados a MPI o de librerías de MPI más adecuadas a ethernet, etc

Hemos propuesto diversas mejoras en todos los ámbitos analizados encaminadas a la reducción de latencia y mejora del comportamiento del tráfico HPC para entornos de convivencia con otro tipo de flujos.

Finalmente se ha desarrollado un pequeño piloto en el que se han puesto en práctica algunos de los mecanismos que contribuyen a la convivencia de aplicaciones en este tipo de entornos heterogéneos obteniéndose mejoras en el comportamiento de tráficos HPC y no HPC en convivencia.

Del estudio realizado y de la implementación de este piloto hemos podido mostrar que en los escenarios descritos es relevante la revisión de los puntos que se describen en los siguientes subapartados:

- Debe tenerse en cuenta todos los niveles de la pila OSI de enlace a aplicación para poder determinar los elementos que pueden producir incremento de latencia con el fin de poder actuar sobre éstos.
- Es necesario un cuidado diseño de la infraestructura de red teniendo en cuenta:
    - Elección de los conmutadores evaluando detenidamente su arquitectura interna así como la conectividad de los distintos anfitriones a los diferentes puertos según los usos de las máquinas virtuales que éste pueda alojar.
    - Revisión de los mecanismos de virtualización.
    - Revisión las arquitecturas de las tarjetas de red y de su configuración
    - Revisión de la planificación del alojamiento de las VMs en función de sus patrones de comunicación ya que podría ser un posible criterio que beneficiaría al despliegue de HPC en convivencia.
- La aplicación de mecanismos de calidad de servicio tradicionales es ventajosa y económica, ya que está ampliamente adoptada en conmutadores y sistemas operativos.
- En muchos casos los diseños y usos tradicionales de red nos llevan a tratar de obtener la máxima utilización del ancho de banda de las líneas disponibles. Sin

embargo, cuando se trata de minimizar la latencia eso puede ser contraproducente, porque deriva en la maximización de la ocupación de los búferes, agregando por tanto latencias adicionales. Es por tanto relevante también el análisis los tipos patrones de tráfico de las aplicaciones HPC de cara a determinar aquellas que su ejecución es posible en este tipo de entornos.

- Existen diversos puntos donde, en estos momentos, no es posible la priorización de tráfico o esta priorización es todavía un campo de investigación como es la provisión de servicios diferenciados en entornos virtualizados para los diversos *domU*.
- Durante las pruebas del piloto se identificaron puntos donde no fue posible la aplicación de medidas de QoS como son las colas de entrada de los conmutadores y de las tarjetas de red, sin descartar que existan otros conmutadores en el mercado que sí las implementen o implementaciones de QoS en los servidores en las cuales esto sea posible.
- Unos de los puntos que se ha presentado como crítico en relación a la contención aparecida entre los diferentes tráficos de la infraestructura es la tarjeta de red. Una buena solución sería disgregar el tráfico prioritario y no prioritario en dos NICs separadas, con lo que también se eliminaría problemas de contención en el conmutador.

El problema de convivencia de flujos HPC en entornos *cloud* multipropósito implica generalmente que dicha convivencia se realizará haciendo uso de tecnologías de red más económicas como pueden ser 1 *gigabit ethernet*. No se puede pretender alcanzar las prestaciones de las infraestructuras específicas de altas prestaciones, con sus latencias en orden de nanosegundos, pero el objetivo es obtener unos retardos razonables que permitan la ejecución de tipologías de trabajos cuyas restricciones sean algo menos estrictas aumentando de esta forma la eficiencia del sistema con la compartición de infraestructura para diferentes propósitos.

8.2 Líneas futuras

Adelantamos una serie de líneas de trabajo que conviene revisar de cara a la posible implantación de servicios HPC en entornos *cloud* multipropósito.

Como ya indicamos anteriormente, dada la extensión del presente trabajo, no es posible cuantificar la influencia concreta en la mejora de la latencia o el jitter derivada de unas u otras optimizaciones, por lo que ha sido necesario limitarse a la enumeración de los posibles puntos de mejora y la prueba de aquellos de más fácil verificación. Es necesario sin embargo un análisis detallado de todas las áreas que permita cuantificar el posible beneficio de cada una de éstas mejoras (u otras que puedan aparecer) en relación a la reducción de latencia, perjuicio para los flujos no HPC y su coste asociado.

El piloto de pruebas, por el mismo motivo ya indicado ha sido muy sencillo y se ha limitado a la ejecución de pruebas con tráfico sintético, siendo importante ampliación del mismo en riqueza de casos e incorporando pruebas con tráfico de competencia real y con la ejecución de *benchmarks* naturales.

No se ha entrado en el análisis de los mecanismos de conmutación basados en *mpls*. Su análisis es relevante ya que permite la conmutación automática entre paths *mpls* lo que nos

puede servir para evitar puntos congestionados del CPD y los tiempos de conmutación podrían ser menores (ya que la conmutación se realiza en base a etiquetas, no siendo necesario el procesado de tablas L2). Tampoco se ha tenido en cuenta el posible impacto de protocolos existentes en redes LAN como *spanning tree* cuyos tiempos de convergencia podría ser problemático.

Como se ha comentado existe diversa literatura de investigación cubriendo diversos aspectos de esta temática, siendo necesario un análisis más detallado de los mismos habiendo diversos campos de investigación abiertos en relación a las aplicaciones de ultra baja latencia, HPC sobre entornos cloud y diferenciación de servicios en virtualización.

9 Bibliografía

[01] Alizadeh, M., Kabbani, A., Edsall, T., Prabhakar, B., & Vahdat, A. (2012). *Less is More: Trading a Little Bandwidth for UItra-Low Latency in the Data Center.* Retrieved from http://cseweb.ucsd.edu/~vahdat/papers/hull-nsdi12.pdf

[02] Bientinesi, P., Iakymchuk, R., & Napper, J. (2010). *HPC on Competitive Cloud Resources.* Retrieved from http://www.aices.rwth-aachen.de:8080/~pauldj/pubs/Cloud-TR.pdf

[03] Boxman, J. (2005). *A Practical Guide to Linux Traffic Control.* Retrieved from http://edseek.com/~jasonb/articles/traffic_shaping/index.html

[04] Brown, M. A. (2006). *Traffic Control HOWTO.* Retrieved from http://linux-ip.net/articles/Traffic-Control-HOWTO/

[05] Chao, H. J., & Liu, B. (2007). *High Performance Switches and Routers.*

[06], C. (2011). *Citrix XenServer Design:Designing XenServer Network Configurations.* Retrieved from http://support.citrix.com/servlet/KbServlet/download/27046-102-666250/XS-design-network_advanced.pdf

[07] Constantinos Evangelinos, C. N. (2008). *Cloud Computing for parallel Scienti?c HPC Applications: Feasibility of running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2.* Retrieved from http://www.cca08.org/papers/Paper34-Chris-Hill.pdf

[08] Das, S., & Sankar, R. (2012). *Broadcom smart-buffer technology in data center switches for cost-effective performance scaling of cloud applications.* Retrieved 09 01, 2012, from http://www.broadcom.com/collateral/etp/SBT-ETP100.pdf

[09] David Newman, C. (2008). *"Design Best Practices for Latency Optimization".* Retrieved from http://www.cisco.com/application/pdf/en/us/guest/netsol/ns407/c654/ccmigration_09186a008091d542.pdf

[10], D. (2006). *"Press Release".* Retrieved from "Press Release": http://www.force10networks.com/news/pressreleases/2006/pr-2006-03-27a.asp

[11] Ekanayake, J., & Fox, G. (2009). *High performance parallel computing with clouds and cloud technologies.* Retrieved from http://grids.ucs.indiana.edu/ptliupages/publications/cloud_handbook_final-with-diagrams.pdf

[12], E. (2012). *Congestion Management and Buffering in Data Center Networks.* Retrieved from http://www.extremenetworks.com/libraries/whitepapers/WPCongestionManagementandBuffering_1856.pdf

[13] Goglin, B. (2012). *Design and implementation of Open-MX: High performance passing over generic ethernet hardware.* Retrieved from http://hal.archives-ouvertes.fr/docs/00/21/07/04/PDF/article.pdf

[14] Hillenbrand, M. (2011). *Towards Virtual Infiniband Clusters with Networks and Performance Isolation.* Retrieved from http://os.ibds.kit.edu/downloads/da_2011_hillenbrand-marius_virtual-infiniband-clusters.pdf

[15] Iakymchuk, R., Napper, J., & Bientinesi, P. (2010). *Underutilizing Resources for HPC on Clouds.* Retrieved from http://www.aices.rwth-aachen.de:8080/~iakymchuk/pub/AICES-2010-06-01.pdf

[16] Jackson, K. R., Ramakrishnan, L., Muriki, K., Canon, S., Cholia, S., Shalf, J., et al. (2010). *Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud.* Retrieved from http://www.lbl.gov/cs/CSnews/cloudcomBP.pdf

[17] Jeff Shafer, P. W. (2007). *Optimizing Network Virtualization in Xen.* Retrieved from http://www.research.ibm.com/haifa/Workshops/systor2007/present/11-Willy-Zwaenepoel.pdf

[18] Jeyakumar, V., Alizadeh, M., Màzieres, D., Prabhakar, B., & Kim, C. (2012). *EyeQ: Practical Network Performance Isolation for Multi-tenant Cloud.* Retrieved 09 01, 2012, from https://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&ved=0CF0QFjAA&url=https%3A%2F%2Fwww.usenix.org%2Fsystem%2Ffiles%2Fconference%2Fhotcloud12%2Fhotcloud12-final38_update6-26-12.pdf&ei=EhU0ULeALYSxhAf2hoCYBw&usg=AFQjCNFTSxyishHThY8wh_2WA

[19], J. (2010). *EX Series Ethernet Switches: QoS-Enabling the Enterprise.* Retrieved from http://www.juniper.net/us/en/local/pdf/whitepapers/2000255-en.pdf

[20], J. (2010). *EX/QFX Series Platform and Technical Overview.* Retrieved from http://www.juniper5daagse.nl/wp-content/uploads/2012/02/1-SSEX01B.pdf

[21], J. (2008, Julio). *NetworkWorld.* Retrieved Agosto 31, 2012, from http://www.commsolutions.com/uploads/NetworkWorldEXCoverage.pdf

[22], J. (2010). *Virtual Chassis Performance: Juniper Networks EX Series Ethernet Switches.* Retrieved from http://www.juniper.net/in/en/local/pdf/merckx62010050400.pdf

[23], J. (2010). *Enabling class of service on EX series switches in a campus LAN.* Retrieved 09 01, 2012, from http://www.juniper.net/us/en/local/pdf/implementation-guides/8010073-en.pdf

[24], J. (2012, Mayo). *Juniper Networks EX8200 Virtual Chassis Performance and Scale.* Retrieved Agosto 31, 2012, from http://www.juniper.net/us/en/local/pdf/industry-reports/2000474-en.pdf

[25], J. (2011, Febrero). *www.juniper.net (Products & Services, Switching, QFabric Family, QFX3500, Literature, Industry Reports).* Retrieved 08 31, 2012, from http://www.juniper.net/us/en/products-services/switching/qfx-series/qfx3500/#literature

[26] Kamal, H., Penoff, B., & Wagner, A. (n.d.). *SCTP versus TCP for MPI.* Retrieved from http://www.cs.ubc.ca/labs/dsg/mpi-sctp/sc2005sctp_mpi.pdf

[27] Lucas Nussbaum, F. A. (n.d.). *Linux-based virtualization for HPC clusters.* Retrieved from http://kernel.org/doc/ols/2009/ols2009-pages-221-234.pdf

[28] Mittal, S. (2009). Retrieved from Differentiated Network QoS in Xen: http://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&ved=0CCEQFjAA&url=http%3A%2F%2Fwww.cse.iitb.ac.in%2Fsynerg%2Flib%2Fexe%2Ffetch.php%3Fid%3Dpublic%3Astudents%3Asaransh%3Astart%26cache%3Dcache%26media%3Dpublic%3Astudents%3Asaransh%3Arep

[29] Prakash, P., Dixit, A., Charlie Hu, Y., & Kompella, R. (2012). *The TCP Outcast Problem: Exposing Unfairness in Data Center Networks.* Retrieved from https://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=6&cad=rja&ved=0CHEQFjAF&url=https%3A%2F%2Fwww.usenix.org%2Fsystem%2Ffiles%2Fconference%2Fnsdi12%2Fnsdi12-final126.pdf&ei=aAk1UP3xJMeZhQf17oCYDQ&usg=AFQjCNHxiov3MJWtq5CjMbpNQx09gJBduQ&sig2=WXqU39

[30], Q. (2011). *Introduction to Ethernet Latency.* Retrieved from http://www.qlogic.com/Resources/Documents/TechnologyBriefs/Adapters/Tech_Brief_Introduction_to_Ethernet_Latency.pdf

[31] Willmann, P. (2008). *A 10 gigabit programmable network interface card. How NICs work.* Retrieved from http://www.ece.rice.edu/~willmann/teng_nics_hownicswork.html

Anexo

Configuración de QoS en conmutadores Juniper EX para la priorización de tráfico HPC

```
operator@JunEX_CLOUD# show class-of-service
classifiers {
    ieee-802.1 hpc_ba {
        forwarding-class hpc1 {
            loss-priority low code-points af11;
        }
        forwarding-class hpc2 {
            loss-priority low code-points af12;
        }
    }
}
forwarding-classes {
    class hpc1 queue-num 5;
    class hpc2 queue-num 6;
}
interfaces {
    ge-0/0/18 {
        scheduler-map puerto-acceso-sched;
    }
    ge-0/0/19 {
        scheduler-map puerto-acceso-sched;
    }
}
scheduler-maps {
    puerto-acceso-sched {
        forwarding-class hpc1 scheduler sp-sched;
        forwarding-class best-effort scheduler sp-sched;
    }
}
schedulers {
    sp-sched {
        priority strict-high;
    }
}
```

Configuración de las clases de servicio en Junos

```
operator@JunEX_CLOUD# show firewall family ethernet-switching filter clasificador-multicampo
term hpc1-mf {
    from {
        source-address {
            10.112.1.0/24;
        }
    }
    then {
        forwarding-class hpc1;
        loss-priority low;
    }
}
term default {
    then {
        forwarding-class best-effort;
        loss-priority high;
    }
}
```

Filtros para clasificación de tráfico (VLAN 1001 HPC y VLAN 1002 para Servicios)

```
operator@JunEX_CLOUD# show interfaces ge-0/0/18
description "Puerto maquina server 1";
mtu 9216;
unit 0 {
    family ethernet-switching {
        port-mode trunk;
        vlan {
            members [ Red1-34 Red_1001 Red_1002 ];
        }
        native-vlan-id RedCloud;
        filter {
            input clasificador-multicampo;
        }
    }
}
```

Aplicación de la clase de servicio en el interfaz del servidor 1

```
operator@JunEX_CLOUD# show interfaces ge-0/0/19
description "Puerto maquina 10-1 PFM David";
mtu 9216;
unit 0 {
    family ethernet-switching {
        port-mode trunk;
        vlan {
            members [ Red1-34 Red_1001 Red_1002 ];
        }
        native-vlan-id RedCloud;
        filter {
            input clasificador-multicampo;
        }
    }
}
```

Aplicación de la clase de servicio en el interfaz del servidor 2

Configuración de QoS en Linux para la implementación de tres colas de QoS

```
# Priorización mediante colas pfifo
tc qdisc add dev eth1 root handle 1: prio bands 3 priomap 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1;
tc qdisc add dev eth1 parent 1:1 handle 11: pfifo;
tc qdisc add dev eth1 parent 1:2 handle 12: pfifo;
tc qdisc add dev eth1 parent 1:3 handle 13: pfifo;

tc filter add dev eth1 protocol ip parent 1: prio 1 u32  match ip dst 10.112.1.0/24 flowid 1:1;
tc filter add dev eth1 protocol ip parent 1: prio 1 u32  match ip dst 10.112.2.0/24 flowid 1:2;
tc filter add dev eth1 protocol ip parent 1: prio 1 u32  match ip dst 0.0.0.0/0 flowid 1:3;

# Priorización mediante colas sqf
tc qdisc add dev eth1 root handle 1: prio bands 3 priomap 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1;
tc qdisc add dev eth1 parent 1:1 handle 11: sfq;
tc qdisc add dev eth1 parent 1:2 handle 12: sfq;
tc qdisc add dev eth1 parent 1:3 handle 13: sfq;

tc filter add dev eth1 protocol ip parent 1: prio 1 u32  match ip dst 10.112.1.0/24 flowid 1:1;
tc filter add dev eth1 protocol ip parent 1: prio 1 u32  match ip dst 10.112.2.0/24 flowid 1:2;
tc filter add dev eth1 protocol ip parent 1: prio 1 u32  match ip dst 0.0.0.0/0 flowid 1:3;



# Priorización mediante tbf
tc qdisc add dev eth1 root handle 1: prio bands 3 priomap 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1;
tc qdisc add dev eth1 parent 1:1 handle 11: htb default 10;
tc class add dev eth1 parent 11:0 classid 11:10 htb rate 100000000kbps ceil 100000000kbps burst 0 cburst 0;
tc qdisc add dev eth1 parent 1:2 handle 12: htb default 10;
tc class add dev eth1 parent 12:0 classid 12:10 htb rate 500000kbps ceil 100000000kbps burst 0 cburst 0;
tc qdisc add dev eth1 parent 1:3 handle 13: pfifo;

tc filter add dev eth1 protocol ip parent 1: prio 1 u32  match ip dst 10.112.1.0/24 flowid 1:1;
tc filter add dev eth1 protocol ip parent 1: prio 1 u32  match ip dst 10.112.2.0/24 flowid 1:2;
tc filter add dev eth1 protocol ip parent 1: prio 1 u32  match ip dst 0.0.0.0/0 flowid 1:3;
```