# A Heuristic Interrupt Coalescing Approach for Improving High Performance Network

Huijun Wu[1] and Jigang Wang[2]
[1]School of Software, Shanghai Jiao Tong University, China 200240
[2]Strategy Planning, ZTE Corporation, China 610041

***Abstract*—With the rapid development of cloud computing technology, high performance network has been widely deployed in commercial cloud computing centers. This gives rise to the challenge of how to deal with the heavy network flow between virtual machines staying on different physical servers. SR-IOV (Single root I/O Virtualization) technology is a new solution to this problem based on hardware assistance principle. However, present SR-IOV technology still face interrupt coalescing challenges when there are multiple virtual machines running on the same physical server. Our paper first demonstrates that the throughput and CPU utilization has not achieved its best when the server is under multi-VM environment. Then we make analysis on this phenomenon and give out a heuristic algorithm to optimize the interrupt coalescing process of SR-IOV technology. Our evaluation result shows that our algorithm can improve inter-server network performance by up to 62.97% and reduce CPU resources consumption by up to 13.16% under certain typical circumstances.**

***Keywords- high performance network; network virtualization; SR-IOV; interrupt coalescing; heuristic algorithm***

## I. INTRODUCTION

In present day's commercial cloud computing environment, high performance networking interface such as 10-Gigabit Ethernet (10GE), has become an essential part in facing the challenge of increasingly heavy network flow between numerous virtual machines inhabiting different physical servers, especially when the cluster environment has the need to achieve highly efficient and scalable I/O Virtualization.

Single Root I/O Virtualization (SR-IOV) thus derived from original direct I/O techniques to help eliminates the overhead from redundant data copies and the virtual network switch in traditional network handling process through the support of hardware virtualization assistance. However, original SRIOV techniques does not perform its best in situation where redundant interrupts are unexpectedly generated due to un-scalable interrupt coalescing model which causes unnecessary CPU overhead and the decrease of network throughput under different circumstances.

In this paper, we study the network performance pattern under changing cluster environment using SR-IOV with 10GE networking interface. We identify the bottleneck of high performance network as the interrupt coalescing mechanism and raise an effective heuristic algorithm which applies adaptive interrupt rate control on the 10GE networking interface. This heuristic algorithm is based on a dynamic approach to adjust the current interrupt throttle rate to the best suitable value according to the changing network environment conditions.

The rest of the paper will be organized as followings. We will first introduce some related work regarding the techniques to improve the performance of I/O virtualization. Then we will discuss and make analysis on the phenomenon and network performance pattern when there are multiple virtual machines under different interrupt throttling value. After that we will show our design of a heuristic algorithm which optimizes the network performance based on our previous analysis. And finally, we will show you the evaluation and its results which measure CPU utilization and the inter-server throughput and compare them with default interrupt coalescing mechanism's results.

## II. RELATED WORK

### A. Interrupt Handling Process Optimization

Typically, there are two ways to optimize the interrupt handling process when dealing with high performance networking environment. One is through traditional interrupt coalescing methods, one is through hybrid method.

- Researchers like Mogul first recognize that when network flow is heavy, too frequent interrupt handling will cause live lock on the receiver side. They raise their suggestions to add priority level to different interrupts and they think that occasional polling is needed to avoid live lock problem.[1]
- Dovrolis and his fellows make analysis on the increased cost of interrupt handling under modern super-scalar processor environment. They raise a suggestion to dynamically adjust the polling internals based on the packet receiving rates. [2]
- Salim and his fellows introduce a hybrid interrupt coalescing mechanism into Linux kernel which is known as NAPI mechanism. This mechanism provide interfaces to drivers of network devices and under the cooperation of these devices, the interrupt mitigation is achieved. [3]
- Salah and his fellows make analysis on the advantages and disadvantages when using interrupt coalescing technology on the high performance network interface. They came to the conclusion that too frequent

interrupt coalescing will result in the destruction of stable TCP flow. They raise an approximate model to explain the relationship between interrupt coalescing delay and TCP throughput stability. [4].

### B. *Network I/O Virtualization Optimization*

There are also many researchers focusing on the network I/O Virtualization optimization.

- Ram and his fellows raise the technique of virtual machine device queues which is abbreviated as VMDq technology. This technology enables the network interface to provide each virtual machine its unique queue and eliminates the cost of packet distribution and copy work in VMM. [5]
- Landau and his fellows believes the traditional trap and emulate model is key bottleneck when we want to improve the network performance under heavy network flow. They then raise a split execution model in which the interrupt will only be sent to the physical core on which the VMM is running on. [6]

## III. NETWORK PERFORMANCE PATTERN ANALYSIS

We conduct our network performance pattern exploration experiment under real high performance network environment.

The metrics we are concentrating at are the interrupt throttle rate and the corresponding throughput between three VMs inhabiting two independent physical servers. These two servers are connected with each other through a network wire with the bandwidth of 10Gbps to avoid hardware's restriction on the performance of the network between these two physical machines. The packet size is 15000 bytes. The Figure I illustrates the experiment result.

From Figure I we can see that the throughput first increases at a very rapid speed as the interrupt throttle rate increase. This is because when we have not use up all the CPU resources on the packet receiver side, the increase on the interrupt throttle rate will allow for more interrupts coming in one second. This will result in more timely collection of coming packets and will result in low delay of response time. However, as the interrupt throttle rate come to the number of 48000HZ, the throughput reaches its peak in the whole experiment. After that, the throughput will decrease as the interrupt throttle rate goes up. This is because we have use up all the CPU resources available and as we receives more interrupts further, we will spend most of our precious CPU cycles on the interrupt handling process instead of handling the packets and data.

From our analysis above, we can conclude that if the number of interrupts exceeds a certain value which is highly affected by the changing network environment, redundant interrupts will be generated and cause loss on the performance of network. And there is a most suitable interrupt throttle rate for changing network environments which enable us to achieve the best throughput and avoid waste of precious CPU resources.

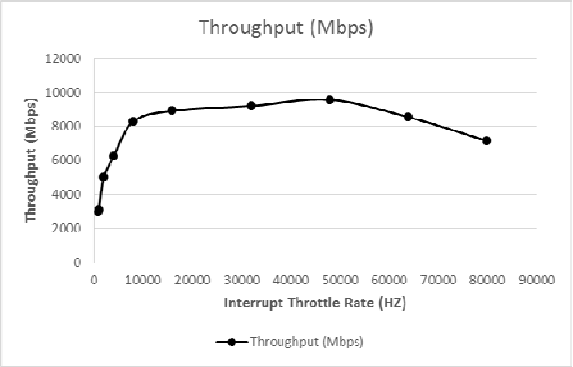


FIGURE I. IMPACT OF INTERRUPT THROTTLE RATE ON NETWORK THROUGHPUT

## IV. HEURISTIC INTERRUPT COALESCING ALGORITHM

Our solution to the problem we illustrated above is to design a heuristic interrupt coalescing algorithm which will dynamically adjust the interrupt throttle rate and try to approach the optimized value which will lead to the best throughput in a high performance network environment. Our algorithm is illustrated as following:

HEURISTIC INTERRUPT COALESCING ALGORITHM

```
Init:
Low=MinITR,High=MaxITR
CutRate=0.618 //Use Golden Ratio as the cut rate
Begin:
Internal=High-Low
while Internal>AcceptInternal do
    TestLow=Low+(1-CutRate)*Internal
    TestHigh=Low+CutRate*Internal
    ThptLow=GetThroughput(TestLow)
    ThptHigh=GetThroughput(TestHigh)
    if ThptLow< ThptHigh then
        High=TestHigh
    else
        Low=TestLow
    end if
    Internal=High-Low
end while
ASSERT(Internal<AcceptInternal)
BestITR=(Low+High)/2
Return BestITR
End
```

Instead of using a mathematical model to calculate out the best suitable interrupt throttling rate, we adopt a heuristic search method to approach the most optimized value in a more dynamic way. This is because in real network environment, there will be various VMs providing different service on a single physical server, thus we may not be able to predict and figure out the precise mathematical model behind the scene.

Apart from what has been discussed above, we should notice that there will be many interrupts happening in one second, and if we consume too much CPU resource on the algorithm, we will not be able to achieve our goal to saves CPU cycles from redundant interrupts and improve the network performance. Thus our heuristic search method is a more reasonable solution.

Our algorithm works by repeatedly excluding ranges that the most optimized interrupt throttling value cannot exists. MinITR and MaxITR is determined by the hardware which defines the initial lower bound and upper bound of interrupt throttling rate. GetThroughput is a sub-function which will measure the instantaneous throughput based on statistics collected by the network interface driver. This function give indication of how to shrink the range and finally make the variable Low and High both approach the best suitable value.

In our algorithm, we will test the throughput in a very short time slice to evaluate the actual effect of our last interrupt throttle rate modification. Since the environment is changing, thus the best interrupt throttle rate may vary as well so we adopt the design to restart the heuristic algorithm every 1000 interrupts.

This algorithm also has the feature of robustness when it is deployed under various network conditions. Within its iterations, it will constantly verify the actual performance result of last interrupt throttling modification and exclude intervals where the best interrupt throttle rate cannot exists. Since it does not rely on specific network environment parameters, it does not have algorithm parameters to be configured before deployed in actual environment. This feature can be ascribed to the algorithm's heuristic style and also make it scalable when applied to servers running unknown number of virtual machines.

## V. PERFORMANCE EVALUATION

In this section, we will give out our evaluation method and result about the effect of the heuristic interrupt coalescing algorithm. We will demonstrate the algorithm's effect by comparing its experiment result with the default interrupt coalescing mechanism which are both carried out under the same real network environment.

In our evaluation, we mainly focus on the CPU consumption and the throughput between two interacting physical servers. We will first explain our evaluation setup and then we will show the evaluation result and make analysis on them.

### *A. Evaluation Setup*

We conduct our evaluation on two physical servers. We make one physical server as the evaluation server and we make another physical server as the evaluation client. The evaluation server and the evaluation client have the same hardware condition. Both of them are Dell PowerEdge R630 Blade servers equipped with Intel E5-2699 2.3GHz CPU. The memory of each physical server is 24GB and each of server has been assigned an Intel 82599 10 Gigabit Ethernet controller. These two servers are connected with each other through a 10Gbps network wire directly.

For the software setup, both the evaluation server and evaluation client's operating system are CentOS 6. On the evaluation server side, multiple VMs is running based on KVM hypervisor. Each of the VM running on the evaluation server has 512 MB memory and has 2 virtual cores. The operating system of the VM is CentOS 6 with the least software installed to avoid the interference on CPU resources consumption measurement. Netperf is installed on each of the VM and Linux's evaluation tool "top" is installed on the evaluation server to collects the statistics result of CPU utilization.

In our evaluation we will use the netperf as the evaluation tool to measure the throughput between two physical servers. And "top" is used to help calculate out the ultimate CPU resources consumption by all VM during the packet receiving time. We configure the packet size as 512 bytes and we conducts experiment from 1 VM to 15 VM to demonstrate the performance tendency in a clearer way.

Figure II shows the throughput evaluation result. Figure III shows the average CPU resources consumption comparison based on 5 duplicate evaluations and Figure IV shows the median evaluation result based on the same set of experiment.

### *B. Evaluation Result Analysis*

As can be seen from Figure II, the network throughput improves evidently by at least 7.56% and at most 62.97% when we adopt the heuristic interrupt coalescing algorithm compared to default interrupt coalescing mechanism.

This is because our algorithm can dynamically adjust the interrupt throttling rate to the best suitable value which enables the full speed of packets handling and at the same time avoid too frequent interrupt coming. This process will eliminate the redundant interrupts, thus reserving CPU resources for handling packets instead of wasting precious CPU cycles in the interrupt handling process.

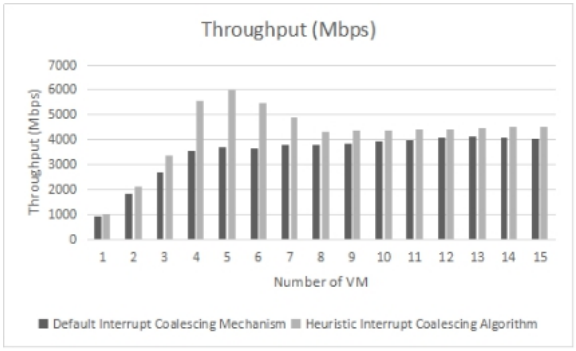


FIGURE II. IMPACT OF INTERRUPT THROTTLE RATE ON NETWORK THROUGHPUT

From Figure III and Figure IV, we can see that with the increase of virtual machine numbers, the CPU utilization will first increase in a comparatively rapid speed and then the increase will become mild.

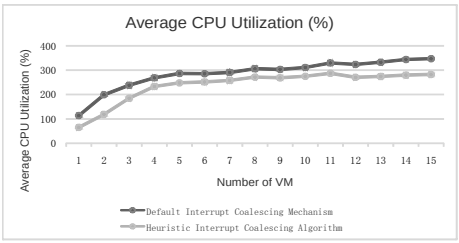


FIGURE III. AVERAGE CPU UTILIZATION WHEN RECEIVING PACKETS ON THE EVALUATION SERVER

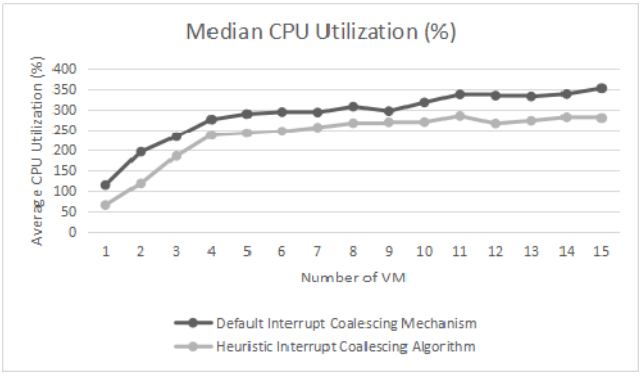


FIGURE IV. MEDIAN CPU UTILIZATION WHEN RECEIVING PACKETS ON THE EVALUATION SERVER

This is because the number of physical CPU is limited for multiple VM to share and after the VM number achieves certain level it will access the ceiling and will not be able to consume more CPU resources any more.

However, when given specific VM numbers, our algorithm can enable the evaluation server to consume less CPU resources by more optimized interrupt throttle rate control. Part of the saved CPU resources was used to complete the actual packets receiving work load and others part of the saving of CPU resources is reflected in the figure. The evaluation result shows that our algorithm can saves at most 13.16% of CPU resources compared to default interrupt throttle mechanism.

## VI. SUMMARY

In this paper we make analysis on the network performance pattern. And based on our analysis result, we design and implement a heuristic interrupt coalescing algorithm which eliminated unnecessary interrupts in high performance network environment. We evaluate the performance of our algorithm and find that it can improve the network throughput by at most 62.97% and saves CPU resources by at most 13.16%


## ACKNOWLEDGMENT

I want to extend my most sincere gratitude to all the people who help me along the way. They are my fellows in the labs who use their spare time to give valuable advices to my work. They are my families and my close friends who support my life and work along the way. Without these people and their encouragement, I may endure much greater hardship in my research work.


## REFERENCES


[1] Mogul J C, Ramakrishnan K K. Eliminating receive livelock in an interrupt-driven kernel[J]. ACM Transactions on Computer Systems, 1997, 15(3): 217-252.

[2] Dovrolis C, Thayer B, Ramanathan P. HIP: hybrid interrupt-polling for the network interface[J]. ACM SIGOPS Operating Systems Review, 2001, 35(4): 50-60.

[3] Information on https://en.wikipedia.org/wiki/New_API

[4] Salah K. Integrated performance evaluating criterion for selecting between interrupt coalescing and normal interruption[J]. International Journal of High Performance Computing and Networking, 2005, 3(5-6): 434-445.

[5] Ram K K, Santos J R, Turner Y. Redesigning Xen's memory sharing mechanism for safe and efficient I/O virtualization[C]//Proceedings of the 2nd conference on I/O virtualization. USENIX Association, 2010: 1-1.

[6] Landau A, Ben-Yehuda M, Gordon A. SplitX: Split Guest/Hypervisor Execution on Multi-Core[C]//WIOV. 2011.