2014 IEEE International Conference on High Performance Computing and Communications (HPCC), 2014 IEEE 6th International Symposium on Cyberspace Safety and Security (CSS) and 2014 IEEE 11th International Conference on Embedded Software and Systems (ICESS)

Performance Characterisation and Evaluation of WRF Model on Cloud and HPC Architectures

S. P. T. Krishnan*, Bharadwaj Veeravalli† Vetharenian Hari Krishna†, Wu Chia Sheng†,
*Institute for Infocomm Research,
Agency for Science, Technology and Research, Singapore 138632
Email: krishnan@i2r.a-star.edu.sg
†Department of Electrical and Computer Engineering
National University of Singapore, Singapore 117583
Email: {elebv@nus.edu.sg, hkv@nus.edu.sg, wuchiasheng@nus.edu.sg}

***Abstract*—Weather forecasting is a computationally intensive problem to handle as well as an important application problem where Cloud Computing can play a key transformational role. Traditionally the problem of handling such workloads have been considered mainly on High Performance Computing (HPC) architectures owing to their tightly-coupled nature and promising speed-up gains. However, with modern day Cloud platforms offering a scalable service infrastructure, current HPC domain experts continue to deliberate on the usage of Cloud Computing platforms as an alternate cost effective and efficient way of procuring just in time computing. At the same time, there are a variety of factors to be considered before HPC workloads such as weather forecasting can be moved to a Cloud infrastructure. In this paper, we have studied the performance characteristics of Weather Research and Forecasting (WRF) algorithm on three different computing architectures. The architectures include two public domain Cloud service providers and a virtual machine (VM) based setup. Through detailed experiments, we have performed a rigorous comparative analysis of the performance of WRF in each of these architectures and in that process we have identified the strengths and weaknesses of each architecture for such HPC workloads. We believe our work of comparing across 3 different architectures is the first of its kind and could be seen as a pilot reference useful to domain experts and system administrators who are considering to employ Cloud Computing for specific HPC problems.**

I. Introduction

The Weather Research and Forecasting (WRF) model is a mesoscale numerical weather prediction system used in both atmospheric research as well as operational forecasting. It features two dynamical cores, a data assimilation system, and a software architecture allowing for parallel computation and system extensibility [1]. A recent HPC profiling [2] study showed that majority of WRF runtime is spent on MPI_Bcast. MPI_Bcast is classified as a collective communication operation. A collective communication operation is defined as a communication that involves a group of processes. Hence the performance of WRF is greatly dependent on the network topology interconnecting the compute nodes.

In this paper, we conduct a large-scale study with WRF algorithm using three architectures. The three architectures are carefully chosen to emulate real-world HPC deployments and feature three very different hardware configurations comprising of distinct computing and network architectures. The problem size has been chosen such that we are able to scale the deployment with respect to number of compute nodes used during an execution run. Two of these architectures are of Beowulf type clusters [3] and are provided by public Cloud Service Providers that any one can use. One of these Cloud service provider has specifically tuned the deployment for HPC class workloads and features high-speed network interconnectivity. The third architecture is a traditional design comprising of a single multi-core machine that hosts multiple VMs. The uniqueness of this third architecture is that the inter compute node communications is completely virtualised between VMs and does not need to go down the Network Interface Card (NIC) and physical network, which is often the bottleneck, to reach the other compute nodes.

There are two objectives in this study. The primary objective is to perform a comparative analysis of WRF on three different architectures and study its performance characteristics. The interconnection technology and the communication between intra-node and inter-node are analysed on how they affect WRF performance. We believe that this evaluation will be useful for a HPC application developer to identify opportunities to run WRF (and by extension any similar HPC application) to obtain better performance at lowest possible cost. We show that by using various optimisation techniques, performance improvements can be obtained.

The secondary objective is to design a process to identify the optimum number of compute nodes for a given problem size such that maximum performance is obtained without suffering over-decomposition overhead. This goal is achieved by running WRF on a reconfigurable computing platform that allows us to quickly scale up/down the number of nodes used until the optimum number of compute nodes is identified. This approach provides a HPC system administrator greater insight into the number of compute nodes required for a particular HPC algorithm to run at best performance and at lowest possible cost. When scaling up the number of CPU nodes, it is essential to keep track of the tradeoffs involved. Due to the amount of messaging involved, it is better to add CPUs that have a higher concentration of cores versus those that fragment their cores into different or seperate sockets.

This paper is organised as follows: In section II, we examine some related works. In section III we discuss the expirements and discuss the results in section IV. Section V concludes the paper and lists our future plan.

978-1-4799-6123-8/14 $31.00 © 2014 IEEE
DOI

II. Related Work

In this section we would like to highlight two relevant performance evaluation studies of HPC software applications conducted by Walker et. al. [4] in 2008, and Jackson et. al in [5]. The focus of these studies have been on profiling the characteristics of the software applications or the performance of computing infrastructures in terms of network latency, bandwidth and computational speed. There are notable shortcomings.

Jackson et. al. [5] in their study used standard Elastic Compute Cloud (EC2) instances instead of the cluster compute optimised instances. Standard EC2 instances does not have high bandwidth low latency networks and therefore the results should not be compared with results from a HPC system. Second, no comparative analysis was done between a HPC architecture and Amazon Web Services (AWS) EC2. Walker et. al. [4] work did not target any specific HPC application and provides a generic benchmark of the EC2 architecture.

Marathe et. al. [6] in their paper have performed comparative performance analysis of Amazon EC2 versus traditional HPC Clusters. They have also developed pricing methods to set node-hour pricing of traditional HPC Clusters vs Amazon EC2.

Their work however does not feature the usage of any specific HPC application and instead uses several benchmarks from popular suites. Hence, their performance analysis may not be directly applicable to HPC applications.

Vecchiola et. al. [7] in their paper have explored the possibility of using Cloud Service providers as an alternative to science computing grids to perform scientific computing tasks.

Their work is not per se a comparative performance analysis and was meant to be an explorative study for reference purposes only.

Our work is novel in two ways. First, by using a specific real-life HPC application, WRF algorithm in our case, our results are directly useful to the weather and climate research community. Second, we have compared the performance characteristics of a HPC algorithm using three different architectures. We believe the various architectures we have experimented with will help in decision making that help to seek balance between minimizing cost and accelerating performance while retaining flexibility.

III. Experiments

In this section, we explain the set of large-scale experiments that we have performed in this study. We will describe the experimental setup, why it is important to study that architecture and how we did the experiments. In section IV we will delve into the results, analyze them and examine the trends.

A. Traditional Beowulf Cluster

The first architecture that we used to study WRF performance characteristics is the traditional Beowulf [3] cluster. We choose this architecture because we are using a similar setup to predict the daily weather in Singapore. For our experiments, we deployed a 32 node computing cluster at Singtel Alatum [8] Cloud service. All the compute nodes were located within a single Internet Protocol (IP) subnet hence no network traffic routing is required.

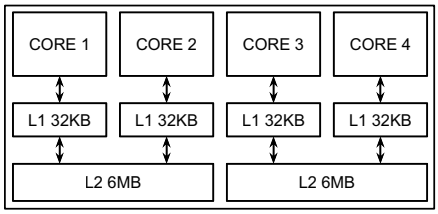

Fig. 1: Singtel Alatum Node Architecture

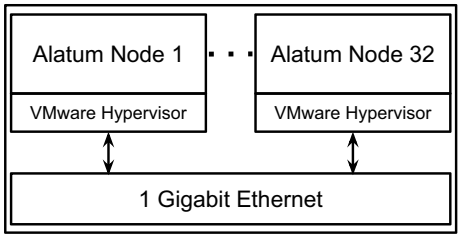

Fig. 2: Singtel Alatum Network Topology

Each compute node contains a dual quad-core Intel Xeon E5430 [9] processor clocked at 2.66 GHz. Each node also contains 15GB of Random Access Memory (RAM). We installed the Linux distribution Ubuntu [10] in each node. The Ubuntu version used was Long Term Support (LTS) 10.04. We then compiled WRF [1] 3.4.1 with Intel fortran compiler [11] and executed using MPICH2 [12] communication protocol.

Figure 1 shows the Central Processing Unit (CPU) architecture of each compute node. We observed that this CPU features a unique CPU cache design. Specifically the L2 cache is split into two units of 6MB and each unit serves two cores. This means that if two threads execute in core 1 and 3, then the Level 1 and Level 2 caches of both cache tree should be flushed and refreshed at every update. We will discuss more about this design's impact for Input/Output (IO)_bound HPC applications in section IV.

Figure 2 shows the network topology of this experimental setup. It can be observed that this is a standard setup of a Cloud Computing architecture. There are two points that need to be highlighted. First, each compute node runs a hypervisor, VMware [13] in this case. All hypervisors incur additional computational costs and by extension more computational time when used with applications that run on a Guest OS; Guest OS runs on top of a hypervisor. This is the same case with VMware as well and the situation is worsened if the HPC application on the guest OS is IO_bound and WRF falls into this category. The second observation is that the network is shared amongst all the nodes in a star architecture. This means that two compute nodes should wait for the ethernet network to be free before they can begin communicating thereby increasing the network latency due to IO_WAIT.

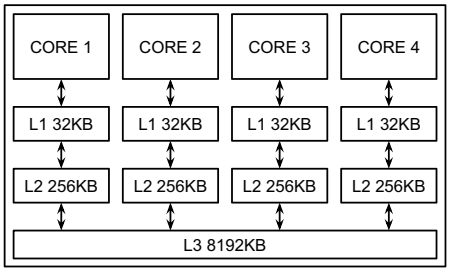

Fig. 3: Amazon Web Services Cluster Compute Node Architecture

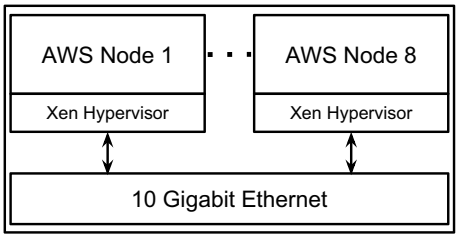

Fig. 4: Amazon Web Services Cluster Compute Network Topology

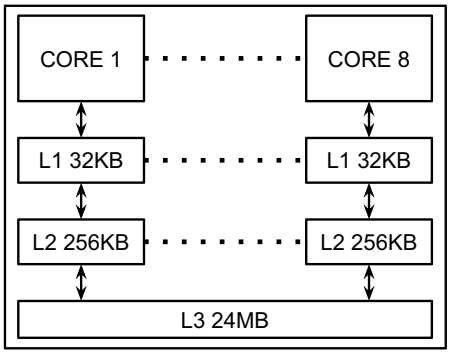

Fig. 5: Multi-core VM Host Node Architecture

B. Amazon Web Services

The second architecture that we used to study the performance characteristics of WRF is the cluster compute product line from Amazon Web Services [14]. We choose this architecture specifically for two reasons. First, it is very close in design to a Beowulf cluster and therefore no application re-design is required for deployment. Second, since the cluster compute product line from AWS is targeted at the high performance computing application space we expected that we are likely to see significant performance gains.

In this experiment we deployed an 8-node compute cluster, each node being of type CC1.4xlarge. CC1.4xlarge is an instance type known as Cluster Compute Quadruple Extra Large. Each CC1.4xlarge contains dual quad-core Intel Xeon X5570 [15] 64 bit Nehalem processors, and has 23GB of RAM. Xen hypervisor is used to run customer workload that contains Guest OS and application. Figure 3 shows the node architecture of the CC1.4xlarge instance type. We observe two differences in this Intel CPU design when compared with the Intel CPU provided by SingTel Alatum. First, there are three levels of CPU cache. Second, the Level 3 cache is shared among all the CPU cores in the same CPU socket.

We specifically used 8 nodes as this simulates our existing in-house setup which also uses 8 compute nodes. The compute nodes are interconnected with 10 Gbps bandwidth Ethernet and were placed in a single placement group. Each placement group offers high-bandwidth and low-latency networking between all nodes within the placement group and this is essential in our experiment since it involves time-critical and high-frequency messages exchanged between the compute nodes. Figure 4 shows the network topology of our experiment in AWS.

We installed the Linux distribution Ubuntu with version LTS 12.04 in each of these nodes. We specifically used a newer version in order to observe any performance gains. We compiled WRF 3.4.1 with Intel Fortran compiler and executed with MPICH2 communication protocol as well. The experiments were setup to run up to 16 processes on a single compute node and we gradually scaled out to run in 8 parallel compute nodes yielding a total of 128 parallel processes. Later, the experiments were repeated by using CPU core pinning option for the hydra (Messaging Passing Interface) MPI manager. We document both the best case of using only physical cores and worst case using a combination of physical/virtual core pairs.

C. Virtualised Infrastructure

The third architecture that we selected to study the performance characteristics of WRF is a single system that is capable of running multiple virtual machines. Our motivation for studying this architecture is to observe the speedup or slowdown when the networking layer is completely virtualised. Specifically, in both the previous architectures the communication traffic between two compute nodes should travel down the physical network stack and back up again to reach the peer VMs.

In this experiment, we used a 32 core physical machine. Each core is an Intel Xeon E7-4830 [16] processor clocked at 2.13 GHz. Figure 5 shows the CPU architecture of this compute node. It can be seen that this Intel CPU architecture is similar to the Intel CPU architecture deployed at AWS. The main difference being a larger (3x) L3 cache. This larger size means that there is less CPU flushing/refiling.

We installed a total of 32 virtual machines and assigned one physical core to each of the VMs. Each virtual machine was assigned 4 GB of RAM and we used static assignment, which means the RAMs are allocated before use. We used the Linux distribution Ubuntu 12.04 as well in each node. WRF 3.4.1 is compiled with Intel Fortran compiler and executed with MPICH2 communication protocol. Each VM had a VM-host local IP address and the VM-host itself acts as a router of network traffic. This means that the inter-VM traffic does not leave the physical machine. Figure 6 shows the virtualised architecture.

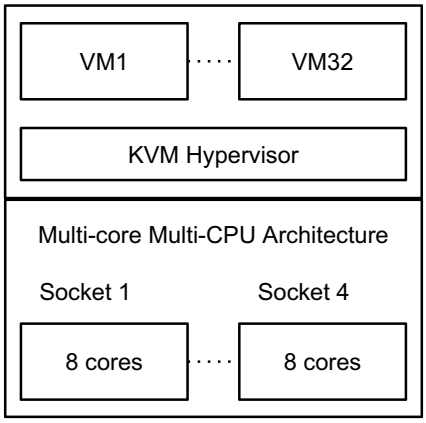

Fig. 6: Multi-core VM Host With Multiple VMs

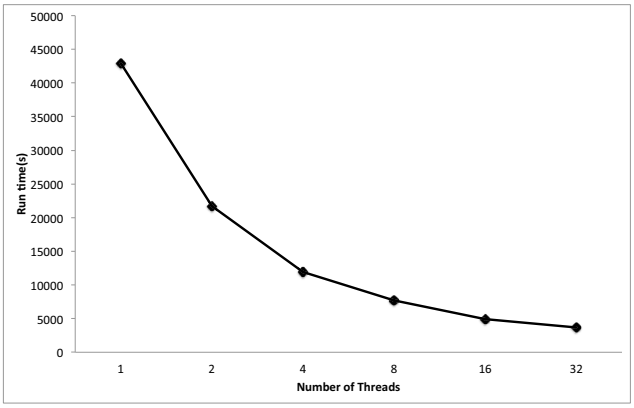

Fig. 7: WRF runtimes using VMs on a Single VM Host

IV. Results & Discussion

In this section, we will first list and discuss the WRF results obtained in the various platforms independently. Following this, we will do a comparative analysis of the results across the three architectures and discuss the reasons we believe are the probable root cause for the visualised performance characteristics.

A. WRF Performance on a Virtualised Infrastructure

Figure 7 shows the runtimes obtained for our WRF experiments using a fixed-size dataset with the number of processes scaled up linearly. The results are in line with what is expected when more parallel compute power is available for the same fixed-size problem that is parallelisable. There are few subtle trends that needs to be highlighted. First, each VM was instantiated with one virtual CPU and pinned to one physical CPU core. We specifically used this 1:1 ratio to ensure that there is no virtual CPU context switching which would happen if more than one virtual CPU is assigned to the same physical CPU or if CPU pinning is not explicitly specified. Second, since each VM is only allocated one virtual CPU, we also ensure that the WRF program only instantiates one process in the VM. This further guaranteed that there is no context-switching among various virtual processes within a single VM. We advocate these two methods as best practices when a virtual infrastructure is being used to solve HPC problems.

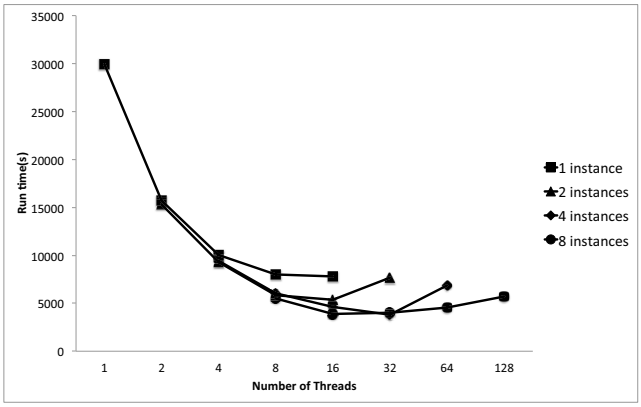

Fig. 8: WRF runtime with multiple threads across multiple instances on AWS

B. WRF Performance on AWS

Figure 8 shows the runtimes for our WRF experiments using a fixed-size data with the number of processes and instances scaled up linearly in AWS. There are four performance lines due to the system configuration differences. In the virtualised infrastructure experiments, each VM had only one virtual CPU whereas in AWS each VM has 8 virtual cores; in actual there are 16 cores but 8 of them are hyper-threaded. This means that it is possible to either run 8 processes in one VM or run 1 process per VM in 8 virtual machines. We wanted to investigate if there is any performance difference between the two configurations and hence we performed all the experimental combinations.

We see a clear trend from Figure 8. The best performance is obtained with 8 instances and each running 4 WRF processes pinned to each of the virtual logical CPUs (and not virtual hyper-threaded CPUs) in each of the instance. We see that the performance degrades when we instantiate more processes per instance. The best performance is also very similar to the performance obtained via the virtualised infrastructure with the key difference being that we only used 8 instances here whereas we used 32 instances in the virtualised infrastructure.

While we were conducting our experiments in AWS, specifically varying the combinations of processes/instances and number of instances, we had to stop and restart the AWS instances several times. As the WRF application is configured to communicate based on IP address, we had to change the configuration file every time. We began to observe that the AWS instances started to get different IP addresses during every reboot. Moreover, these IP addresses were in different subnets as well. We recorded these IP addresses to see if there is some trend. Figure 9 shows the variations in the number of instances per subnet and across fifteen days when we conducted these experiments. Overall, AWS allocated our 8 instances in 11 subnets and during any one run there were

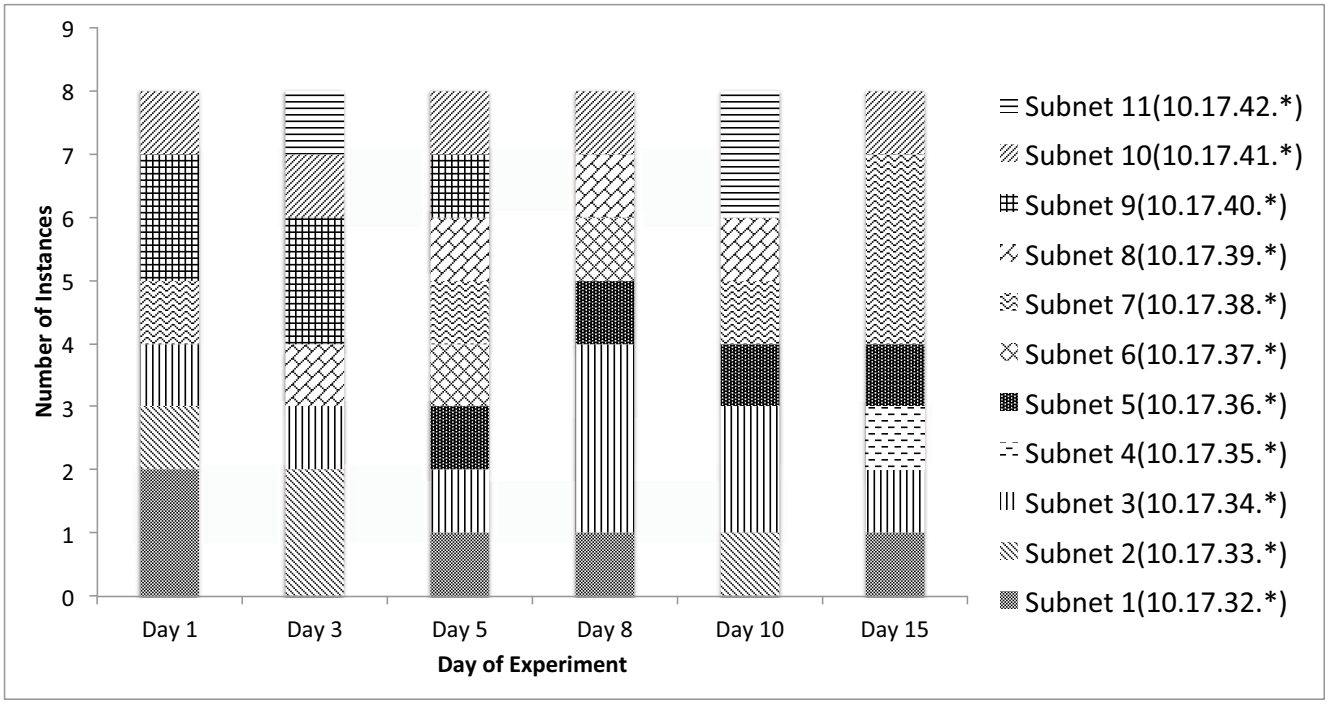

Fig. 9: IP allocation Trends for AWS Instances

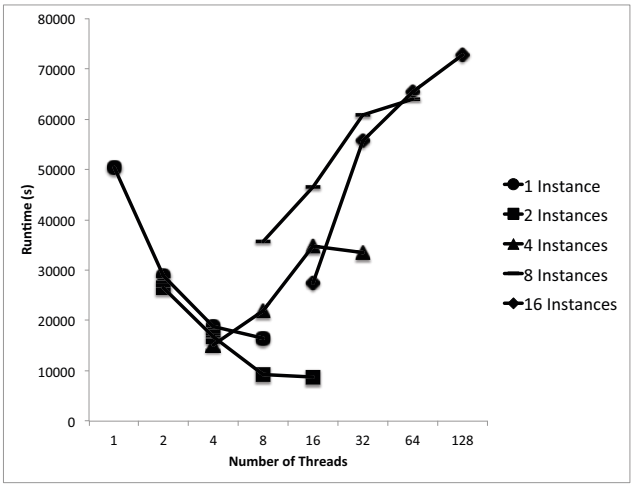

Fig. 10: WRF runtimes with multiple process and instances on Traditional Beowulf Cluster

between 5 - 8 subnets in active use. When instances in two different subnets have to communicate, a gateway node and router needs to be involved to translate the communication between them. This incurs two additional network hops. For a HPC application that is IO_bound this means that every data packet has two additional hops at the least and introduces additional network latency although a small value.

C. WRF Performance on Traditional Beowulf Cluster

Figure 10 shows the WRF runtimes using the same fixed-size WRF problem set and using variable number of processes and compute nodes with the nodes interconnected using a traditional beowulf type cluster. Needless to say that the performance characteristic of WRF in this network is very different from those of AWS and virtualised infrastructure. Several trends can be observed at the microscopic level and we detail it below.

WRF performance seems to increase with increase in the number of parallel processes while using one instance. In this case, no application-generated communication traffic is expected to be at the network layer. Similar performance characteristic can be seen for the curve that depicts the runtimes using two instances. We restricted the number of processes to 8 per instance because of the number of physical processors available in a compute node.

WRF performance degrades significantly when the number of instances is more than two. We observed, as the number of parallel processes is increased along with number of the compute instances, instead of seeing better performance we actually witnessed performance degradation. The trend remains consistent throughout the experiment after this point. We attribute this observation to the fact that there are more than one pair of processes distributed across compute nodes, which are ready to communicate, and is competing for the availability of the network resources. The competition increases with more process pairs and since only one pair is able to communicate at one point in time, others are forced to enter IO_WAIT thereby increasing the CPU clock time for the WRF software.

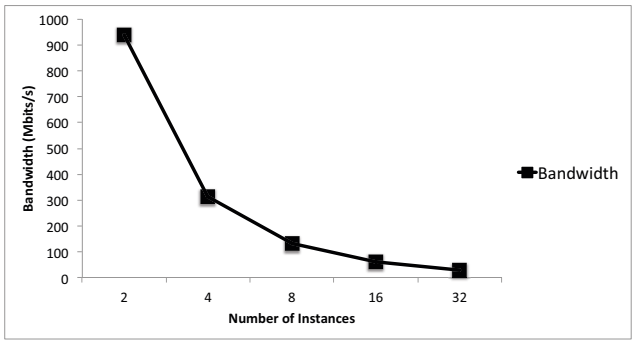

Fig. 11: Bandwidth Performance of Singtel Alatum Service

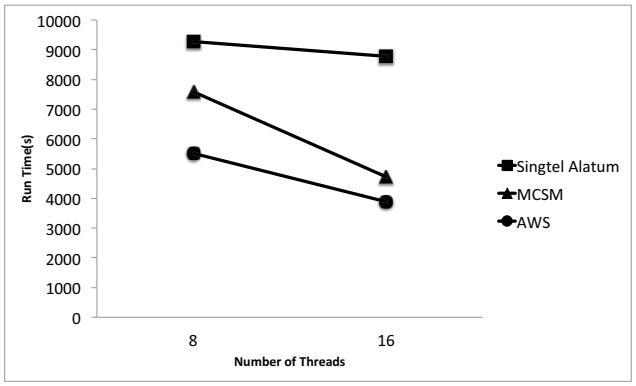

Fig. 12: Comparison of best case runtimes across three architectures

In order to understand the impact of this communication problem we undertook a network bandwidth / load test study. IPerf [17] tool from Linux OS was used for generating the network traffic. First, we used one pair of instances in which one node generated network traffic and the other consumed the network traffic. Subsequently, we doubled the number of pairs of instances until we were able to use all the 32 instances that was available to us. Figure 11 shows the results of this bandwidth and load test. It can be observed that the network performance drop to a third from 2 to 4 instances and with 16 pairs of producers & consumers running across 32 nodes, the effective network bandwidth to each of them is in the range of 30 Mbps. For HPC workloads that are IO_bound like WRF software this acts as a severe bottleneck.

D. WRF Performance across all three Architectures

In this section, we perform comparative analysis of WRF software across the three architectures using the runtimes observed from each of the experiment. We highlight these comparative insights below.

First, from the results in previous sections we see that WRF runtimes generally improve as the number of processes increases in all the three architectures, where there is a dedicated compute core per process. The runtime trend inverses in both AWS and SingTel Alatum when the number of simultaneous processes is more than the number of dedicated compute cores. In case of AWS this degradation happens when the number of processes used to run WRF exceeded 32 processes. The runtime performance in Singtel Alatum was adversely degraded when the number of instances used was greater than 8.

Second, in the case of AWS we observed a slight improvement in runtime performance when number of instances used to spawn the processes is greater than the same number of processes. An example, referring to Figure 8, would be the case where 8 threads are being executed in 1/2/4/8 instances. It can be seen that the runtimes are better when the number of instances is more for the same number of threads. We attribute this observation to the possibility of physical CPU being shared by multiple processes running in the same VM. Since AWS is a public Cloud, we don't have root / administrator access to the VM host unlike in our in-house virtualised infrastructure experiment. Therefore, it is possible that the VMs may not have been pinned to different physical CPU and could either be free floating or shared with fewer physical CPUs.

Third, comparing the runtimes between AWS and SingTel Alatum, we observe that the AWS cluster compute placement group has played a key role in scaling up the performance while having more instances to run together. Conversely, Singtel Alatum, which does not offer common placement group, suffers high overhead cost in communication between instances as the number of threads scale up. Specifically, as the number of thread increases, the bandwidth available to them becomes a dominant factor over the processing power of each machine. This is largely due to the increase in communication necessary to partition tasks effectively across multiple processes.

Four, for cases where it is possible to do 1:1 scheduling (such as one process to one compute core or one VM to one physical core) compute nodes that have less physical cores is likely to have generally-slower inter-node communication. This is because the communication now involves a physical network, which is generally slower than intra-node communication that is usually based on shared memory. Conversely, when the number of threads spawned is more than the number of physical cores in a node, CPU context switching results in leading to a performance drop. This can be observed when we instantiate more threads in the case of AWS.

Finally, we compare the performance results of the best configuration from each of these three configurations and show the same in Figure 12. From this figure, it can be seen that the AWS cluster compute gives the best performance and the slope of the performance curve is gradual between 8 and 16 threads. In comparison, the performance curve is steeper for the second best output for the virtual infrastructure. In comparison, the best case scenario from SingTel Alatum is distinctively slower. More importantly, the scaling up results in much poorer performance as well.

E. Optimisation Techniques

In this section, we highlight the optimisation techniques that we used in some of our experiments and also propose new ones to further enhance the performance observed.

1) Compilation flags: It is highly recommended that HPC applications be compiled from source either on the target system or using a cross-compiler that is aware of the target architecture. In our case, we compiled WRF software from the source code that we downloaded from National Centre for Atmospheric Research [1] on each of the systems.

In addition to the compilation process, we also recommended that the HPC application developer use chip-specific compilation flags for optimisation instead of using architecture-wide compilation options. This ensures that the best possible performance is obtained through the usage of chip-specific compiler flags.

2) Network bandwidth: For IO_bound applications, it is important that there is sufficient network bandwidth available for inter-node communications. Most HPC applications are IO_heavy and will need low-latency high network bandwidth regularly even if they are CPU_bound by design.

In our study, we found that WRF is an IO_bound app and therefore requires comparatively more bandwidth compared to CPU_bound applications. Through our experiments, we found that a bandwidth of even 1 Gbps was not sufficient and a higher bandwidth is recommended for better performance. For example, AWS provides a mechanism known as placement group which ensures that the instances assigned to that placement group are assigned a low-latency 10 Gbps bandwidth network.

3) IP Subnet: In addition to the high-bandwidth requirement for IO_bound HPC applications, it is also important to reduce the number of network hops when two compute nodes are communicating. If the two compute nodes are in two different IP subnets, then an IP address translation is required through the use of a network router and this may introduced unnecessary delays and may be a potential bottleneck as well.

In our experiments with the virtualised infrastructure, we used IPs that belong to the same subnet. However, such an option was not available for us in our experiments with AWS. Hence it is recommended that public cloud providers like AWS ensure that all IPs are within the same subnet in order to minimise the number of network hops.

4) Socket clustering: CPU pinning is a very important method which ensures that the process is tied to a specific CPU core without being shuffled around by the CPU scheduler as part of the scheduling process. We employed this technique successfully and generally observed better performance compared to the case when no such technique was used.

In addition to core pinning, we would like to propose a new method that we called socket clustering. A CPU socket is the physical CPU that is added to the motherboard. CPU designs vary in the number of physical and virtual core per socket. For HPC applications that use co-operative processes along with shared memory objects, it would be optimum if the individual processes are pinned to the core within the same socket. This will ensure that the processes implicitly use the multi-level caches effectively.

5) Non-Uniform Memory Access (NUMA) vs Uniform Memory Access (UMA): We also recommend the exploration of NUMA [18] architecture for HPC applications and observe the performance variance against their UMA [19] counterparts.

V. Conclusion & Future work

In this paper, we have conducted a large-scale experiment to study the performance characteristics of the WRF algorithm; WRF software is the gold standard used by meteorologists worldwide to study weather and climate patterns. Our interest in initiating this study was to explore newer architecture types that could yield better performance compared to the traditional beowulf clusters. To this end, we used two Cloud Computing providers & one in-house system and repeated the same experiment using the same input dataset in each of them. We reported the results from these experiments and also did a comparative analysis across them. We concluded this study by noting a few key observations. First, newer Cloud Computing products such as cluster compute instances from AWS is promising for HPC workloads from both deployment and cost-effectiveness perspectives. Second, on the compute side we identify twin factors that yield better performance i.e., the ratio of active processes to the available compute cores and the pinning of these processes to the CPU cores. CPU pinning plays two key roles. It ensures that each process (application or VM) has a dedicated hardware resource and the ill effects of cache flush/refill is also minimised. On the network layer, we concluded that the interconnection topology between instances plays a key role. This is evident from the better performances that we observed with both AWS and virtual infrastructure. Hence, we believe it is important to focus on the high-bandwidth, high-speed and dedicated network otherwise any computation performance gained by having more instances to run HPC applications might be overshadowed by the interconnection latency, communication overhead and decomposition overhead.

We plan to extend this work by using light-weight application containers such as docker [20] instead of the heavy-weight hypervisors in our virtualised infrastructure and observe the performance characteristics. We believe this will be interesting and promising from a performance perspective since a hypervisor simulates a hardware while a container emulates a OS kernel.

VI. Acknowledgements

The first and second authors would like to thank the following organisations for their support - Infocomm Development Authority (IDA) [21], Singapore for providing the SingTel Alatum cloud resources and Uber Cloud HPC Experiment [22] for providing part of the AWS cloud resources. All authors would also like to thank TMSI [23], Singapore, for supplying required data for the experiments carried out in this study and the SingTel Alatum technical support team for their resource provisioning assistance to carry out our experiments. Finally, the authors would like to thank Mr. Yian Chee Hoo of Institute for Infocomm Research for proof-reading the manuscript and suggesting important & relevant changes to the same.

References

[1] National Centre for Atmospheric Research, "Weather Research Forecasting model," http://www.wrf-model.org/index.php.

[2] H. A. Council, "Weather research and forecasting performance benchmark and profiling," HPC Advisory Council, Tech. Rep., 2012.

[3] "Beowulf Cluster," http://en.wikipedia.org/wiki/Beowulf_cluster.

[4] E. Walker, "Benchmarking amazon ec2 for high-performance scientific computing," *USENIX Login*, vol. 33, no. 5, pp. 18–23, 2008.

[5] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. Wright, "Performance analysis of high performance computing applications on the amazon web services cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, Nov 2010, pp. 159–168.

[6] A. Marathe, R. Harris, D. K. Lowenthal, B. R. de Supinski, B. Rountree, M. Schulz, and X. Yuan, "A comparative study of high-performance computing on the cloud," in *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, ser. HPDC '13. New York, NY, USA: ACM, 2013, pp. 239–250. [Online]. Available: http://doi.acm.org/10.1145/2462902.2462919

[7] C. Vecchiola, S. Pandey, and R. Buyya, "High-Performance Cloud Computing: A View of Scientific Applications," in *Proceedings of the 10th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN 2009)*. IEEE Computer Society, 2009.

[8] Singapore Telecommunications Ltd., "Singtel Alatum," http://www.ngp.org.sg/events/2011/SingTel Alatum Clouds Service.pdf.

[9] Intel Corporation, "Intel Xeon Processor E5430," http://ark.intel.com/products/33081/Intel-Xeon-Processor-E5430-12M-Cache-2_66-GHz-1333-MHz-FSB.

[10] Canonical Ltd., "Ubuntu Server," http://www.ubuntu.com/.

[11] Intel Corporation, "Intel Fortran Compilers," https://software.intel.com/en-us/fortran-compilers#pid-23480-97.

[12] MPICH, "MPICH2," http://www.mpich.org/.

[13] VMware Inc., "The Architecture of VMware ESXi," http://www.vmware.com/files/pdf/ESXi_architecture.pdf.

[14] Amazon.com, Inc., "Amazon Web Services High Performance Computing," https://aws.amazon.com/hpc/.

[15] Intel Corporation, "Intel Xeon Processor X5570," http://ark.intel.com/products/37111/Intel-Xeon-Processor-X5570-8M-Cache-2_93-GHz-6_40-GTs-Intel-QPI.

[16] ——, "Intel Xeon Processor E7-4830," http://ark.intel.com/products/53676/Intel-Xeon-Processor-E7-4830-24M-Cache-2_13-GHz-6_40-GTs-Intel-QPI.

[17] The Iperf team, "Iperf," http://sourceforge.net/projects/iperf/?abmode=1.

[18] "Non-Uinform Memory Access," http://en.wikipedia.org/wiki/Non-uniform_memory_access.

[19] "Uniform Memory Access," http://en.wikipedia.org/wiki/Uniform_memory_access.

[20] Docker, Inc., "Docker," https://www.docker.io/.

[21] Infocomm Development Authority Singapore, "I D A singapore," http://www.ida.gov.sg.

[22] Uber Cloud HPC Experiement, "Uber cloud," http://www.theubercloud.com/hpc-experiment/.

[23] Tropical Marine Science Institute, NUS, "TMSI research labs," http://www.tmsi.nus.edu.sg/.