The evolving grid paradigm and code “tuning” for modern architectures- are the two mutually exclusive?

Robin Long
Physics Department, Lancaster University, Lancashire, U.K., LA1 4YB.
E-mail: r.long@lancs.ac.uk

Abstract. With the data output from the LHC increasing, many of the LHC experiments have made significant improvements to their code to take advantage of modern CPU architecture and the accompanying advanced features. With the grid environment changing to heavily include virtualisation and cloud services, we look at whether these two systems can be compatible, or whether improvements in code are lost through virtualisation. We compare the runtime speed improvements achieved in more recent versions of ATLAS code and see if these improvements hold up on various grid paradigms.

1. Introduction
During 2013-2014 the LHC was upgraded to increase the centre-of-mass energy from 8TeV to 13TeV. This necessitated a need to substantially reduce the CPU time required to process data. The software needed to be able to handle an increase in event rates from 500Hz to 1kHz, along with an increased number of interactions per bunch crossing.

At the same time, the Grid began shifting towards more dynamics setup that could be used to leveridge idle and opportunistic compute time. By using virtual machines on big computing farms such as the HLT (High Level Trigger) which sits idle when no data is being taken, the HLT could be rapidly re-purposed into a Grid site. Whilst such systems are more dynamic and obviously offer increased flexibility, grid resources are strained and an ever increasing amount of data is coming out of the LHC. To deal with these increasing demands, the Grid has looked to places where it can make a more opportunistic use of resources, and the experiments have opted to re-write their software to make it more efficient. Some of these changes are minor, such as changing from 32-bit to 64-bit, however some changes make use of certain other CPU changes such as vectorisation. With these changes being made a comparison is needed to see if the move towards more dynamic resources such as virtualisation is in conflict with software improvements.

In particular we look at two grid paradigms: Containerisation and Virtualisation. In virtualisation the hyper-visor, which controls the virtual machines, takes up a non insignificant amount of resource on the physical node. Containerisation, whilst being more limited has a much lower overhead[ref your other paper here]. This paper therefore examines how the changes made to the ATLAS software to decrease run time are affected by the compute node being containerised using Docker, and virtualised using a KVM virtual machine.
Figure 1. A graphical representation of how Bare Metal, virtualisation (KVM), and containerisation (Docker) works. To run a service on a baremetal machine, we need the kernel, user-land (libraries that interact with the kernel), and the software we want to run. When we create a virtual machine, we need to virtualise all of this. Using containers, we only need to virtualise the user-land and software as the container’s user-land interacts with the host kernel.

2. Virtualisation
Virtualisation has been widely adopted by business to enable multiple operating systems, often of differing types (Linux / Windows), to leverage a single piece of hardware. This has led to large scale reduction of hardware both in terms of servers but also in the connecting infrastructure needed to support such hardware; monitors, keyboards, racks, air-conditioning to name but a few. The ability to run many virtual machines on a single host is attractive in business to save money through hardware but also savings in software purchases. The rapid provisioning of machines due to the removal of the need to buy a physical server for each operating system instance is also very attractive.

KVM is one of the leading Open-source virtualisation solutions; the hyper-visor comes in the form of a service installed on a standard Linux installation. This allows for the virtualisation of hardware, and the freedom to use any operating system as the guest.

3. Containerisation
Traditional virtualisation and para-virtualisation requires the whole operating system and hardware to be virtualised. Using containers, individual applications can be separated from each other and “virtual machines” can be created but without the overhead of traditional setups.

Containers allow the running of individual applications and their dependencies. These, in the case of Docker, then run on top of the Docker services. Only the user-land and applications are virtualised which allows significant overhead savings. This is detailed in Figure 1. As only user-land and applications are virtualised, an operating system that requires a different kernel type such as MS Windows or GNU Hurd cannot be containerised on a Linux host.

For this analysis we use Docker which is the leading containerisation solution on Linux.

4. The testing environment
Our setup consists of two compute nodes running Scientific Linux 6. These are of identical specifications, both having 16 core Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz CPUs, with 64 Gb RAM and a 1TB Western Digital Hard drive.

Configuring the compute nodes to act as a KVM hyper-visor was relatively trivial. The standard KVM packages were installed from the repositories. A single virtual machine utilising all available resources was created using virt-manager.

Configuring the compute node to act as Docker host took a little more work. As it was non trivial to mount CVMFS inside the container, CVMFS was mounted on the host machine and
then added as a volume to the container by using the “-v” flag on the command line. A single container was created which once again utilised all available resources.

The Docker container used was an official Centos 6 container with the necessary grid software installed on top. An external volume called “/data” was used to save analysis output into, and the container was run interactively. The command line options are listed below where “longr/centos6-chep” is the Centos 6 container with grid software installed

Docker -D run -v /data:/data -v /afs:/afs -v /cvmfs/atlas.cern.ch:/cvmfs/atlas.cern.ch -i -t longr/centos6-chep /bin/bash

5. Analysis

Previous analyses have focussed on benchmarking of nodes. However it is important to test real world code, especially as the software is beginning to diverge away from the benchmarking methods. We specifically focus on the ATLAS software stack, which has had major changes made to its code, and has also changed the libraries that it relies on. The changes studied have resulted in a three fold speed up in run time.

In this analysis we look at three different grid node setups: Bare Metal, full virtualisation using KVM, and containerisation using Docker. By running four different releases of the ATLAS software on the 3 different node configurations, we hope to see whether the performance benefits brought about from the software changes are maintained, and whether the two virtualisation techniques have any impact on runtime speeds.

The specific changes made to the code are beyond the scope of this article, and readers are instead referred to the ATLAS publication, ATL-SOFT-PUB-2014-004, for such in-depth discussions. The analysis focus’s directly on whether the change in grid paradigm, that is, the use of virtualisation or containerisation, instead of bare metal has any effect on the speed increases that have been achieved in new versions of the ATLAS software.

Four versions of the ATLAS software were used, 17.2.7.9, 19.0.3.3, 19.1.1.1, and 20.1.2. The biggest changes in reconstruction speed came in the change to 19.0.3.3 where 64bit was used instead of 32bit. After this various improvements were made to the software which are detailed in ATL-SOFT-PUB-2014-004.

The reconstruction code was ran twice, for each software version on 2 nodes. The reconstruction speed listed in the results is the average of 10,000 for each instance. All datasets used were kept on local disk to avoid any network lag.

The results are shown in Figure 2 where two things are immediately noticeable: Firstly, all three methods (Bare Metal, Virtualisation, and containerisation) have very similar levels of improvement as the software version is changed; Secondly, the time taken to reconstruct individual events is comparable for all three methods across the 4 software versions.

As the software versions are improved dramatic decreases are seen in the time taken to reconstruct each event (in milliseconds). These improvements are seen across all three paradigms. Whilst virtualisation and containerisation are slower, there is no noticeable change in the rate of progress as the software is improved.

For Docker and Bare Metal there is a significant amount of overlap in the results. Any performance loss in Docker is within the variation seen in the results. In some cases Docker appears to be faster than Bare Metal, but more repetitions removes this artefact and shows that the two are almost identical.

6. Conclusion

Virtualisation and containerisation are shown to have no significant effect on total reconstruction time. For version 17.2.7.9 of the software, Docker takes on average 0.17 (0.32%) seconds less to reconstruct an event compared to Bare Metal, whereas virtualisation takes 2.5 seconds (4.81%)
Comparison of Reconstruction Speeds  
(ATLAS Preliminary Simulation, RDO to ESD)

![Graph showing comparison of reconstruction speeds]

**Figure 2.** Performance of (a) HEP-SPEC and (b) IPerf for the various machine configurations. Note that neither of these plots start at zero so that the differences can be seen more clearly.

longer. By version 20.2.1.Y this has changed to 0.1 (0.85%) and 0.58 seconds (4.66%) longer respectively.

Overall this analysis shows that whilst virtualisation has significant overheads, that lead to around a 5% increase in reconstruction time, this is consistent as changes are made to the software. Further analysis is required to identify if this is due to the virtualisation of the disk, or the CPU itself. Such a distinction is necessary as reconstruction is just one of many types of software used at the LHC and all vary between being CPU or IO intensive.

Docker shows variation across the analysis. Whilst there is an increase of 0.3% to 0.9% in reconstruction times, in some cases Docker was shown to be slightly faster than Bare Metal. Further analysis is required here as this could either be a lack of statistics, or an improved API for access kernel routines. Either way, no real performance loss is seen. As Docker uses the native disk, albeit via a different filesystem, little IO performance changes are expected, but these need to be verified.

Overall, the improvements being made to ATLAS reconstruction software are not harmed by any of the considered Grid paradigms. Furthermore, the software is not hugely affected by differing paradigms to begin with. It must be stressed that this is one small subset of ATLAS software, and more varied tests across the range of software types need to be conducted.
7. Acknowledgements
The authors would like to thank the ATLAS collaboration for allowing use of the software and
datasets for this analysis. Furthermore a special thanks is extended to Antonio Limosani, whose
help and scripts made this analysis possible.

References
[1] HEP-SPEC06 Benchmark (Accessed 19/05/2015);
https://w3.hepix.org/benchmarks/doku.php
[2] IPerf - TCP and UDP bandwidth performance measurement tool (Accessed 19/05/2015);
http://code.google.com/p/ipertf/
[3] Bonnie++ homepage (Accessed 19/05/2015);
http://www.coker.com.au/bonnie++/
[4] Docker (Accessed 19/05/2015);
https://www.docker.com/
[5] KVM, Kernel-based Virtual Machine
http://www.linux-kvm.org/page/Main_Page
[6] ATLAS offline reconstruction timing improvements for run-2
https://cds.cern.ch/record/1955923/files/ATL-SOFT-PUB-2014-004.pdf