A Performance Evaluation of Sequence Alignment Software in Virtualized Environments

Zachary J. Estrada, Zachary Stephens, Cuong Pham, Zbigniew Kalbarczyk, and Ravishankar K. Iyer
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Email: {zestrad2,zstephe2,pham9,kalbarcz,rkiyer}@illinois.edu

Abstract—The prospect of simpler infrastructure management and affordability has garnered interest in cloud computing from bioinformaticians. However, the performance cost of adopting such an infrastructure model for bioinformatics is not fully known. In an effort to help quantify this performance cost, we ran synthetic benchmarks and measured the runtimes of two short-read alignment applications on cloud-like virtualization environments. The environments were implemented utilizing the KVM hypervisor, the Xen hypervisor, and Linux Containers. We compare the runtime in each environment against a physical server and offer discussion and insights. Though the applications perform similar operations, we observe that their performance characteristics differ, as do their performance in the different virtualized environments. We attribute the differences to the way that these programs utilize system resources. We find that the more CPU-bound Novoalign is much less sensitive to virtualization environments than BWA is, and has near-physics server performance even when virtualized. Additionally, we find that static CPU pinning can improve performance, and we demonstrate that Linux Containers offer performance comparable to that of a physical server.

I. INTRODUCTION

This paper investigates the performance of cloud computing for bioinformatics by measuring the performance of alignment applications across three different VMMs/hypervisors. Cloud computing is moving past hype and becoming mainstream, thanks to the advantages of its utility model over that of traditional data centers. Recent surveys [1], [2] confirm the expanding adoption trends of cloud computing in various application domains. In addition to cost savings, as IT staff are freed from “fire-fighting” infrastructure maintenance, cloud computing can offer additional benefits, including increased efficiency and more time that can be allocated towards innovation [2].

The aforementioned advantages of cloud computing make it an attractive alternative for bioinformaticians and genomics researchers who are facing the inefficiencies of traditional methods of maintaining and delivering computational resources [3]. Recent advances in high-throughput DNA sequencing technology place a heavy burden on computing infrastructure in an effort speed up the overall DNA sequencing process and benefit from the economies of scale. A quicker and less expensive way to produce sequencing data consequently requires more storage and more processing power to accommodate subsequent analyses of the data. Cloud computing becomes a natural solution to address the problem of increasing demand [3], [4], [5], thanks to its affordable price, flexible resource provisioning, and offloaded infrastructure management.

The key enabling technology for cloud computing is server virtualization. Though virtualization has existed in one form or another for many decades [6], recent adoption in commodity server platforms (e.g. x86_64) has made virtualization ubiquitous in today’s data centers. The effort to virtualize has mainly been fueled by growing enterprise IT environments, and the primary benefits of virtualization (isolation, flexibility, and density) help these environments scale more efficiently while cutting costs. Cloud computing further exploits these benefits at a larger scale and uses virtualization to abstract computing resources as a utility model.

Despite its benefits, virtualization introduces nonzero overhead, and thus there is an inherent performance cost for virtualized systems. There are several different approaches to system virtualization, including instruction emulation, paravirtualization, and hardware virtualization (further details are in Section II), each of which offers different management features and performance costs to applications. Furthermore, given a particular implementation of a virtualization technology, various aspects of an application can impact the virtualization performance overhead differently in subtle ways. Thus, understanding the trade-offs and differences between virtualization technologies is important when one is deciding whether or not to move to the cloud, or which cloud platform to choose. In this paper, we investigate three of the most popular virtualization implementations: Linux Kernel Virtual Machines (KVM) [7], Xen [8], and Linux Containers (LXC) [9]. While the kernel features that enable LXC cannot be classified as a VMM or hypervisor in the strictest sense, for the sake of simplicity, we will refer to all three of these technologies as VMMs.

II. BACKGROUND

A. Bioinformatics Workloads and Alignment

The most resource and time intensive step in many Next-Generation Sequencing (NGS) pipelines can be the alignment step, where short reads are mapped to a reference sequence. In collaboration with a major genomics institution,\(^1\) we have

\(^1\)The institution wishes to remain anonymous
studied a production variant calling pipeline. We have observed that alignment is one of the dominant components and can take roughly 30% of the total execution time of the entire workflow. Therefore, when one is looking to reduce costs or gain mobility by moving tasks to a cloud, it is important to consider the effect of doing so on this critical component of many genomics workflows. It can be expected that various alignment programs would utilize computing resources in a similar way, so we run two popular alignment packages to test this hypothesis and derive conclusions that we intend to be representative of alignment programs in general. One is the Burrows-Wheeler Aligner[10] (BWA) and the other is Novoalign[11]. We chose BWA as it is a popular open-source package and Novoalign as it is widely considered by many in the bioinformatics community to be the most accurate alignment program available[12]. From a computer systems perspective, these applications are interesting as they take the same input and work towards a similar objective, but using two different algorithms.

B. Kernel-based Virtual Machines

Hardware assisted virtualization (HAV) technologies provided by different CPU vendors (e.g. VT-x from Intel or SVM from AMD) add extensions to the instruction set to readily support Virtual Machines. These extensions allow for a simpler and potentially more secure hypervisor. The Linux Kernel Virtual Machine (KVM) hypervisor utilizes HAV and serves as a good example for a hardware-assisted virtualization platform.

KVM is a valuable use case as it is a natural choice for Linux VMs since it is included in the upstream Linux kernel. Furthermore, and is widely considered the de facto standard for the open-source Infrastructure-as-a-Service (IaaS) cloud management platform Openstack2 and the virtualization management platform libvirt.

C. Xen

Xen is considered a full Type 1 hypervisor, even the “host” OS that has direct access to the hardware and manages the other VMs runs inside a special VM called “Domain 0.” Xen helped popularize the concept of para-virtualization, where one runs a modified version of the operating system to avoid the unnecessary redundancy and overhead of providing the illusion of a full physical system to the OS.

Xen is an important use case because the most widely used IaaS platform, Amazon EC2, is built on top of a modified version of Xen. While the results presented here may not be representative of performance on EC2, Xen is still a popular hypervisor.

D. Linux Containers

Linux Containers (LXC) represent a different virtualization paradigm that offers less isolation than the previous two VMMs, i.e. all containers share an operating system kernel[13]. The advantage of sharing an OS kernel means that there is no device emulation and no redundancy from running an OS on top of another OS. This method is not virtualization in the usual sense, but containers can be used for many of the same reasons one would use virtual machines. The isolation provided by LXC is stronger than usual OS process isolation, thanks to the use of cgroups[14] and kernel namespaces[15]. Kernel namespaces allow one to have separate domains for kernel objects (i.e. processes, users, network interfaces, and mounts) and cgroups allow one to allocate and partition resources amongst different processes.

Container-based virtualization is gaining popularity and acceptance, as evidenced by the successful open source project Docker.3

III. METHODOLOGY

A. Performance Measurements in Virtual Machines

Since a virtual machine does not have complete control over the hardware, some care should be taken to ensure accurate measurements. One particular problem for performance measurements is unstable timekeeping inside VMs[16]. It is easy to see that since a VM timeshares with host operating system, there will be more clock drift than expected for a physical server. Also, one could expect this drift to be more unpredictable as it is dependent on the state of a live system (e.g. how applications running on the hypervisor and other VMs use the system) compounded with the usual sources of clock instability (temperature fluctuations, etc...). As a simple example, consider a VMM implemented using Intel’s VT-x HAV. When a virtual machine attempts a certain action (e.g. a trusted operation), it induces a VM Exit event. Control is then transferred to the hypervisor, which performs some task related to the action and returns control to the VM. During this switch when the VM is not running a gap could be introduced in the VM’s virtual timestamp counter (TSC). While there are well established techniques for minimizing this in use today (e.g. a para-virtualized clock synchronized with the host, kvm-clock), precise timing measurements in VMs are still difficult to obtain. For precise measurements, it is considered best practice to use a stable reference outside of the VM[17]. While the timing measurements used in this work are of a large enough scale to for these effects to be insignificant, we use a 3rd party server to measure timestamps so that the same clock is used for all measurements. For higher resolution timing, one could use the hypervisor and a trapping mechanism (e.g. hypercall) as opposed to an external server.

When benchmarking applications that use a large amount of data, it is important to be mindful of how these machines cache this data. If one is using virtual machine images that are layered on top of an existing filesystem (as could be expected in a cloud environment utilizing the KVM hypervisor), then both the VM operating system and host operating system could be caching data. Therefore, it is important to clear the buffer cache between all data samples to ensure that each experiment starts in a similar state (one could also reboot the machines to

2http://openstack.org

3https://www.docker.io/
ensure a consistent state, but our experimentation has shown this to be excessive). The buffer cache was cleared before every run of each benchmark or application.

Most modern multi-processor architectures have their memory arranged in a Non-Uniform Memory Access (NUMA) configuration. In a NUMA architecture, different processors/cores communicate with different regions of memory at different speeds, with fastest access to “local” memory and slower access to “non-local” memory. For optimal performance, it is clear that a process should only be scheduled to run on a CPU that has local access to the process’s memory. Most modern OS schedulers are aware of NUMA, but one can still direct the OS to only run a process on a certain set of CPUs by setting the process to have affinity for a set of CPUs or “pinning” it to those CPUs (e.g. by using the taskset command in Linux)[18]. Similarly, most VMMs offer the ability to pin vCPUs to certain physical CPUs. In this study, the virtual machines’ vCPUs were pinned4 to avoid negative effects from non-local memory accesses.

IV. Experimental Setup

A. Hardware and Operating System

All experiments were performed on two Dell PowerEdge R720s with dual-socket 8 core Intel Xeon E5-2660 2.2GHz CPUs (3.0GHz Turbo boost). Data was run on both machines to rule out machine-specific bias. The servers have 128GiB of DDR3-1333MHz memory and 8 SAS magnetic hard drives arranged in a RAID 10 array. The RAID controller used (PERC H710P Mini) has a 1024MiB battery backed cache, with the disk array set to ‘write back’ mode. This server configuration can be considered a typical enterprise class large-memory server one could expect to find in a data center or research lab.

B. Virtual Machines

The standard Linux libvirt5 API and toolchain (e.g. virsh) were used to manage the VMs for all platforms. The CPU used supports Intel VT-x HAV technology with VT-d, which includes support for Extended Page Tables. Unless stated otherwise, each VM was allocated 8GiB of memory and 4 vCPUs.

The operating system used was Ubuntu 12.04 LTS (kernel version 3.5.0-45-generic) and the VM images were based on the 12.04 LTS Ubuntu cloud image.6 This platform was chosen as it is very common in IaaS cloud environments and VM images containing this operating system are available by default from most, if not all, major cloud providers.

For each VMM, we attempt to follow standard “best practices” when it comes to performance tuning, i.e. ensuring all supported cpu features were available to VMs’ vCPUs and using para-virtualized drivers wherever possible (e.g. virtio for KVM and using the para-virtualized version of Xen 4.1.5).

C. Simulated Read Data

All of the results presented below used a copy of the same input dataset, to ensure that differences observed were not caused by variance in input data. The input data consisted of paired-end 75bp reads sampled uniformly from human chromosome 1 (UCSC hg19) with 2x coverage using the ART[19] Illumina read simulator. The simulated FastQ files contained approximately 3 million reads with an average phred score of 22 and mean fragment length of 200bp. We will refer to this dataset as chr1 throughout the remainder of this work.

Since use paired-end reads, this means BWA was run with the aln command twice and the sampe command took the output of each aln to perform the mapping. Both aligners were configured output alignments in the SAM format.

V. Results

This section presents the data that was collected during our experiments as well as specific methods used in each experiment. Analysis and discussion of these results is contained in Section VI.

A. Synthetic Benchmarks

In an effort to characterize the performance of each VMM and provide context for the rest of the results presented in this work, we ran two synthetic benchmarks to learn how different aspects of each VMM performs. We hypothesize that two key areas where VMs would differ from physical servers would be in disk I/O (due to device emulation) and memory access (due to virtualized memory management). We use two standard benchmarks to investigate these differences on our platform. In all of the figures presented, “XEN-pv” is used to clearly state that we are considering the para-virtualized implementation of Xen in this work.

1) Bonnie++: Bonnie++7 is a simple open-source filesystem-level disk benchmarking suite. We use bonnie++ to characterize the disk performance of each platform.

Fig. 1 shows the results of running bonnie++ on each VMM platform. As can be seen, for the machines in our environment, differences in disk I/O performance were not statistically significant for sequential reads or writes. To ensure that the buffer cache does not skew results, bonnie++ recommends using a test file size of at least twice the size of system memory. Following this recommendation involved the physical server using a significantly larger file size (256GiB) compared to what the VMs used (16GiB). This choice most likely explains the seemingly anomalous seek times where the physical server had a slower random seek rate (as it had a large file to seek within) than KVM and LXC.

---

4http://libvirt.org/


6http://cloud-images.ubuntu.com/precise/
2) **STREAM**: STREAM is a standard memory benchmark\(^8\) that is used throughout the computing industry. We ran the STREAM benchmark with an array size of 40M elements, or a total memory requirement of 0.9GiB. The particular CPU has a cache size of 20MiB, which this test exceeded by nearly two orders of magnitude.

The results from running STREAM 30 times (each run itself repeated the benchmark 30 times and reported the best result) are plotted in Fig. 2. Though there are not any substantial differences, the trend seems to be consistent: the physical server and LXC provide the same performance, with KVM followed by Xen.

**B. BWA**

The results of running BWA on all four platforms are shown in Fig. 3. In this test, a paired end alignment with BWA was run 30 times on the chr1 dataset and a mean of the run time was obtained. For this dataset, the application was run single-threaded. Since many sequencing pipelines involve large amounts of data, it is not uncommon for users to run alignment and variant calling applications single threaded and parallelize across data sets (e.g. parallelize across chromosomes when mapping an entire genome). Here, we see that LXC’s performance closely matches that of the physical server, but Xen and KVM have significant overhead. We reiterate that these results are consistent across more runs on both machines.

As a use case more indicative of performance in a shared cloud computing environment, Fig. 4 shows how the runtime changes when 8 virtual machines (or processes for the physical server) are run simultaneously. For this data, we used the same dataset with different copies and multiple VM images. Each run was started at the same time and the completion time reported is when all jobs finished. This scenario was conceived as a “worst case” since all of the VMs should be competing for the same resources at the same time. Here, the slowest task becomes the critical task. This metric was chosen because in many workflows all tasks need to complete before moving on to the next step (e.g. aligning all chromosomes before moving on to whole-genome variant calling).

**C. Novoalign**

Similar to the BWA results presented above, 30 runs of Novoalign were completed and the mean was obtained. Again, these were single threaded computations. It may be helpful to note that the scales on the y-axis of this plot are significantly smaller than with BWA. The only hypervisor comparison that appears to be statistically significant was Xen. Even then, the performance overhead is measured in small single digit percentage points. We attribute this to Novoalign’s CPU-bound nature as it is over two times slower than BWA for our dataset, but reading the same data. Fig. 2 agrees with this observation.

---

\(^8\)http://www.cs.virginia.edu/stream/
as Xen performed worse than other platforms in the STREAM
memory benchmark and is marginally slower here.

As with the previous BWA results, we ran 8 concurrent
Novoalign processes with and without vCPUs pinned. As can
be expected, running more VMs concurrently results in higher
variance.

D. Execution Statistics

In order to better understand the characteristics of each
application, we ran collectl\(^9\) during the execution of a single

\(^9\)http://collectl.sourceforge.net/

instance of each application. Collectl is a utility that collects
OS statistics about a running system such as CPU load,
memory usage, etc... We chose to measure CPU, memory, and
disk usage over time and the outputs from collectl during the
execution of each application are graphed in Figs. 7a and 7b.
Inspection of Fig. 7a confirms with the execution of paired-
end alignments with BAW, the first aln command finished
at roughly 40% of the total runtime and the SAM output is
generated with sampe at roughly 90%, causing the heavy disk
activity. Examining Fig. 7b shows that Novoalign reads the
input files all at once and performs its operations in-memory.

VI. ANALYSIS AND DISCUSSION

Analyzing the data presented earlier, we make the following
observations:

1) **CPU pinning can be effective for managing resource
contention**: One surprising observation was that while modern
OS schedulers are NUMA-aware and attempt to keep pro-
cesses running on the same CPU to avoid cache thrashing, we
were still able to gain a noticeable performance advantage by
statically pinning vCPUs/processes. It should be noted that the
machines were not maximally utilized in our studies and more
intensive workloads would likely exacerbate the performance
loss of poorly placed processes. It appears that for KVM and
Xen, libvirt was able to do quite well by default (it defaults to NUMA-aware placement), but the Linux scheduler
(the default CFS scheduler used by Ubuntu in kernel version
3.5.0-45-generic) did not perform as well as anticipated. Since
that behavior was observed with the Linux scheduler for
regular processes and containers (for Novoalign, the LXC
data are inconclusive on this point for BWA), this observation
about CPU pinning is more general than the cloud computing

\[ 
\begin{array}{c|c|c|c|c|c}
\text{PHY} & \text{PHY-pin} & \text{KVM} & \text{KVM-pin} & \text{XEN-pv} & \text{XEN-pv-pin} \\
\hline
0.95 & 1.00 & 1.05 & 1.10 & 1.15 & 1.20 \\
\hline
\end{array} 
\]

\[ 
\begin{array}{c|c|c|c|c|c}
\text{PHY} & \text{PHY-pin} & \text{KVM} & \text{KVM-pin} & \text{XEN-pv} & \text{XEN-pv-pin} \\
\hline
0.95 & 1.00 & 1.05 & 1.10 & 1.15 & 1.20 \\
\hline
\end{array} 
\]

\[ 
\begin{array}{c|c|c|c|c|c}
\text{LXC} & \text{LXC-pin} \\
\hline
1.12 \\
\hline
\end{array} 
\]
2) Despite having a similar objective, alignment programs can utilize system resources in very different ways: Looking at Figs. 7a and 7b, we can compare how the two alignment applications differ in their use of system resources. It appears that BWA periodically reads from the disk, while Novoalign reads all of its input in one shot. However, we expect that behavior to change with datasets larger than system memory, and further experimentation is required. The applications’ differing behavior provides evidence contrary to our hypothesis in Section II-A that alignment programs should utilize resources similarly. Therefore, users and systems integrators should be careful to observe how their chosen application performs in a given environment.

3) Novoalign is efficient with system resources: Based on continuing observation of how the applications use resources, it appears that while Novoalign takes more than twice the time BWA does to finish execution, it is more efficient with system resources. One must remember that though the objective of these applications is the same, their outputs from the same input data are different (arguments about which aligner produces “more correct” output are beyond the scope of this work). Fig. 7a shows that BWA has a more bursty read/write pattern. It could fill caches and cause the application to block on I/O bandwidth. In Fig. 7b, Novoalign can be seen to make many small writes (as is also evident from the constant steep increase of the buffer cache size). In addition to the battery-backed write cache on these machines’ RAID controllers, modern filesystems (e.g. the ext4 [20] filesystem on these machines) use a delayed allocation cache strategy, and the many small writes that Novoalign performs would likely be more amenable to this strategy than bursts of larger writes.

4) Performance for a given VMM is dependent on the application being executed: In Fig. 3 it is easy to see that KVM performs the worst for BWA, but in Fig. 5, Xen performs worse for Novoalign (although the difference is small). This observation demonstrates that the behavior of one application on a particular VMM is not fully indicative of how another application would behave on the same VMM or how that same application would perform on a different VMM.

5) CPU-bound applications run at near-baremetal speeds in virtualized environments: This was an expected result, but here we have validated it. This is helpful information for bioinformaticians whose workloads are CPU-bound. Furthermore, when considering performance, it appears that as long as a task exhibits this property, the choice of VMM becomes less important. As discussed in VI-2, Novoalign and BWA have different disk access patterns. We believe that Novoalign’s disk access pattern contributes to its having a smaller virtualization performance cost than BWA. That access pattern causes Novoalign to become CPU/memory bound after application startup, and from Fig. 1, it appears that disk seeks are costly in virtualized environments.

6) VMM scheduling overhead is low when VMs are not overprovisioned: From Figs. 4 and 6, we see that all context of this work and applies even to traditional HPC job-scheduling systems.
environments have roughly the same performance costs from resource contention. E.g., a comparison of Fig. 6 to Fig. 5 shows that properly configured environments have an overhead of roughly 5 to 8% in all cases, not just in the virtualized environments. However, many cloud providers over-provision their physical resources (i.e., assign more virtual resources to a physical machine than exist), so future work should investigate scenarios where resource contention is higher.

7) Linux Containers have low performance overhead: Based on our data, it is clear that if one wishes to build a cloud computing environment with performance as the paramount concern, Linux Containers would be the best choice among the options presented here. This result was expected, but the intention of performing the experiments with LXC was to demonstrate and quantify the performance improvement for a bioinformatics workload. However, anyone choosing to adopt a container-based solution should be aware that there are certain drawbacks. We tested with a newer version of libvirt (1.1.1), and the LXC driver still does not support vCPU pinning, which appears to be important for containers based on our experiments. Also, some cloud adopters see the ability to choose one’s entire software stack (including the OS kernel, especially if the users are interested in a particular version or tuning parameters) as an advantage of cloud computing, and it is impossible to do so when using LXC.

VII. RELATED WORK

Cloud computing services have been available for quite some time now, and as the technology has matured, more users have ported their workloads to these environments. As a result, many investigations of how High Performance Computing (HPC) applications perform on different VMMs has been published. The common perception is that the expected performance overhead of ~ 10% is acceptable for these applications when not running tightly coupled MPI-style computations [21]. To put things in perspective, however, in the context of a sequencing workload that could take e.g. 1000 CPU Hours to complete, 10% would be equivalent to roughly 4 CPU days. The authors in [22] ran tightly coupled MPI-based applications and observed significantly degraded performance (6-20x) on EC2. In a study more similar to what was done in this work, the author in [23] used WCD EST, an EST assembler (contrasted with the whole genome assemblers used here) to measure the performance of EC2 for scientific workloads. The author focused on scalability and variability, and the author showed that EC2 can achieve acceptable scalability. This work also highlighted many qualitative benefits of moving to the cloud vs. participating in a typical shared cluster.

A recent study investigated the potential for use of container-based virtualization in HPC [24]. However, that study focused on HPC benchmarks, whereas the work presented in this paper evaluates actual application performance - specific to bioinformatics applications. The authors of that work ran the same memory benchmark, STREAM. That work also demonstrated that LXC and a physical server have near-identical performance, but found that Xen was slower than LXC and the physical server by roughly a factor of 2. It is difficult to determine the cause of the stark difference between their results and those presented in this paper, but the version of Xen was similar (4.1.2 in the other work and 4.1.5 here). However, the operating system used in that study was Ubuntu 10.04 and potentially had less efficient para-virtualization drivers. Nevertheless, our results did agree that Xen’s memory performance is not as efficient as other virtualization methods.

VIII. CONCLUSIONS AND FUTURE WORK

In order to provide meaningful comparisons and keep the work tractable, the vast parameter space of this study was reduced to a few important cases. However, the results presented here are still useful for bioinformaticians who may be interested in adopting a cloud based solution, particularly those who elect to explore the option of a private cloud. As expected, when workloads became more CPU intensive and less I/O bound (e.g. Novoalign as opposed to BWA), performance inside a VM is indistinguishable from a physical server.

As with most performance studies, subtle changes in a system (e.g. CPU affinity) can have a drastic effect on program completion time. More work will need to be done to consider a larger number of parameters than evaluated in this work and a more comprehensive study needs to be conducted before a general conclusion can be drawn. While this study investigated one of the key components of a bioinformatics workload, an interesting future study would be to investigate an entire workflow in the cloud. Furthermore, one could run experiments and compare the performance and financial cost of different public cloud providers for bioinformatics workloads.

IX. ACKNOWLEDGMENTS

The authors are grateful for insightful and helpful discussions with Steven Lumetta, Victor Jongeneel, Liudmila Mainzer, and Gloria Rendon. This material is based on research sponsored by the Air Force Research Laboratory and the Air Force Office of Scientific Research, under agreement number FA8750-11-2-0084. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. All figures in this work were generated using matplotlib [25].

REFERENCES


