Scaleout to Scaleup: From Traditional Supercomputer to NextGen Bigcomputer for Data-intensive Scientific Applications

Arghya Kusum Das, Seung-Jong Park
School of Electrical Engineering and Computer Science
Center for Computation and Technology
Louisisna State University
Baton Rouge, LA, 70801
Email: {adas7, sjpark} @lsu.edu

***Abstract*—The enormous growth of the bigdata produced by different experimental facilities is rapidly changing the model of computation in the domain of high performance computing (HPC). Many HPC aficionados, in order to efficiently manage their data intensive workload started using the current state of the art bigdata analytics softwares like Hdoop, Giraph etc. devieting from the traditional parallel programming models like MPI, Grid etc. However, there is very limited understanding on the performance characteristics of the underlying hardwares that these bigdata analytics softwares can obtain when applied for a data-intensive high-performance scientific workload.**

In this paper we evaluated the performance of three different types of compute-cluster including a traditional supercomputer, called SuperMikeII located in LSU, USA and two private cloud infrastructure, called SwatIII and CeresII located in Samsung, Korea. Our analysis is based upon our own benchmark parallel genome assembler (called PGA) built atop Hadoop and Giraph. The assembly pipeline consists of a huge amount of short read analysis using Hadoop, followed by a large de Bruijn graph analysis using Giraph, thus serving as a very good real world example of data as well as compute intensive workload. We modified the underlying hardware-components and their organization in SwatIII cluster in many different way to evaluate the relative merit(s) of each component individually as well as in terms of the balance among those components. Finally, we concluded the paper after evaluating CeresII, a Samsung microbrick based prototype-cluster with high density servers.

I. Introduction

Scientists in different fields are increasingly handling huge amount of bigdata produced by different experimental facilites which make the so called compute intensive scientific applications a severe data intensive endeavor. Starting from the astronomical data analysis to the coastal simulation, from the social data analysis to the genome assembly, the huge volume of data poses several challneges to the scientific community starting from efficiently storing and managing to optimally processing it. The fundamental model of computation involved in the scientific applications is rapidly changing in order to address these challenges. Deviating from the decade old compute intensive programming paradigm like MPI, Grid etc. many HPC aficionados has started using the current state of the art big data analytics software like Hadoop, Giraph etc. for their data intensive scientific workloads.

Consequently, the traditional supercomputers, even with tera to peta FLOP scale processing power are found to yield suboptimal performance. especially because of the io- and memory-bound nature of the data intensive workloads. The cumulative effect of the CPU, memory, disk and the network on the overall performance of the applications makes the task of providing efficient yet cost-effective hardwares more challenging, however, opens new opportunities for the hardware-manufacturers. Furthermore, in the last few years, a growing number of data-intensive HPC applications started shifting towards the pay-as-you-go cloud infrastructure (eg. Amazon Web Service, Penguin, R-HPC etc.) especially because of the elasticity of resources and reduced setup-time and cost. As a consequence, there is a growing interest in all the three communities, including the HPC-Scientists, the hardware-manufacturers as well as the commercial cloud-service-providers to develop cost-efective, high-performance testbeds that will drive the next generation scientific research involving huge amount of bigdata. Also, millions of dollars are being spent in programs like NSFCloud for the same purpose where several academic organizations and manufacturing companies collaborated to enable the academic research community to develop and experiment with novel cloud architectures.

Despite of this growing interest in both the scientific as well as the industrial community, there is very limited understanding of the performance characteristics of the underlying hardware that the current state-of-the-art bigdata analytics softwares can obtain when applied for high performance data intensive scientific workloads. Thus, we found it extremely important to evaluate different types of distributed cyber infrastructure in the context of a real world data intensive high performance scientific workload.

In this work, we use the large scale de novo genome assembly as one of the most challenging and complex real world example of high performance computing workload that recently made its way to the forefront of bigdata challenges. De novo genome assembly reconstructs the entire genome from fragmented parts called short reads when no reference genome is available. The assembly pipeline consists of huge amount of short read analysis followed by a complex analysis on a largescale graph, thus, serving as a very good example of both data- as well as compute-intensive scientific workload.

Specifically, in this paper, we juxtapose the performance of different distributed cyber infrastructure with our own benchmark large scale parallel genome assembler, called PGA, that we developed using Hadoop and Giraph. We present the performance result of PGA atop three different types of clusters including a traditional-supercomputer, called SuperMikeII located in LSU, USA and two private cloud infrastructure, called SwatIII and CeresII located in Samsung, S.Korea. we always compare the performance-result of any cluster (SwatIII or CeresII) with SuperMikeII as the baseline.

In our study, we evaluate both, the impact of optimizing each hardware component individually (e.g. network-interconnect, storage and memory) as well as the impact of the overall organization of these components in a high performance compute cluster. We always compare the performance-result of any cluster (SwatIII or CeresII) with SuperMikeII, the traditional supercomputer which is considered as the baseline.

1) In the first part, our evaluation started with comparing the impact of different type of network-interconnect (Infiniband vs Ethernet). It is followed by a comparison between different type of storage (i.e. HDD vs SSD) where we found almost 60% improvement in the Hadoop-job than a supercomputing cluster due to increased IOPS and reduced io-wait. It is then followed by evaluating the impact of increasing the amount of memory where we observed almost 35% improvement in the Giraph-job because of the effect of caching.
2) In the second part, we compare several different ways of organizing these hardware-components especially how to leverage SSDs in a cost effective manner in a cloud environment. Starting from a qualitative analysis of various organization, we moved to a semi-quantitative recommendation on the use of SSDs in a scaledout and a scaledup cluster. Finally, we evaluate the performance of CeresII which is a novel prototype-cluster with high-density-servers.

The rest of the paper is organized as follows: Section-II describes the prior works related to our study. In section-III we discuss the programming model offered by Hadoop and Giraph as well as provide a general overview of the expected performance characteristics of traditional supercomputing hardwares. Section-IV discusses the overview of our Parallel Genome Assembler followed by details about the input data in section-V. Section-VI describes different types of cluster archetecture and the Hadoop configurations we used for our evaluation purpose. In section-VII we compare the impact of different network, storage and memory architecture individually with the CPU-utilization, and IO-patterns in details. Finally, in section-VIII we compare different architectural balance in terms of both performance as well as performance/$.

II. Related Work

Earlier studies showed that Hadoop can be useful for data intensive scientific workloads [42]. Consequently, a growing number of codes in several scientific areas such as bioinformatics, geoscience are currently being written using open source state of the art bigdata analytics software like Hadoop, Giraph etc. [24]. Many of the traditional supercomputers also started using myHadoop [24] to provide the scientists an easy interface to configure Hadoop on-demand. However, there is very limited prior work that evaluated different distributed cyber infrastructures for these softwares when applied for data intensive scientific workload. This leaves a fundamental question yet to be answered: *how does a next generation high performance computation cluster should look like to handle data intensive scientific workload*. In this section we provide the related works for our study.

BigData analytics softwares: Hadoop [4] offers a simple, easily scalable disk-based map-reduce abstraction. HBase [6] is a NoSQL-based distributed linearly scalable key-value store targetted the applications that need random, realtime read or write access to tera/peta byte scale data residing in disk. Similarly, Hive [7], Impala [8] etc. are some of the popular disk-based NoSQL Database which provide the users with an SQl like query interface. On the other hand, Piccolo [17] and Redis [16] are two in-memory distributed key-value store, aimed at applications that need low-latency finegrained random access. Giraph [18] is a synchronous, vertex centric, in-memory graph processing framework originated as the open-source counterpart to Google's Pregel [2] that we analyzed in our work. GraphLab [20] is a faster asynchronous graph processing framework mainly motivated to provide the users a framework to write correct machine learning algorithms. Resilient Distributed Datasets (RDDs) [12] in the Spark system, offers a unified in-memory solution for all batch processing, Stream prcessing [13], SQL query [15] and graph processing [21]. Although, the computation model has evolved enough in the last few years to handle data intensive complex scientific workload, the choice of underlying hardware infrastructure still remains a major challenge.

Evaluation of Hadoop for scientific workload on existing superComputers: With growing number of scientific applications written in Hadoop, many different groups studied the performance of Hadoop on different existing supercomputers that they have access to. Jha [43] observed the convergence between traditional HPC and current state of the art bigdata analytics softwares and evaluated both of them in different supercomputing environment with k-means clustering as an example. Fadika [42] studied the performance of Hadoop for common HPC workload namely filter, merge and append. Guo [45] analyzed different graph processing framework with graph500 [36] BFS workload. Although, thse studies provide excellent insights on performance of current state of the art bigdata analytics softwares for different scientific applications, their analysis is confined into the domain of existing supercomputers, thereby, unable to address whether or not we can get better performance in other cyber infrastructure.

Evaluation of Hadoop for enterprise workload on different cyber infrastructure: Several performance analysis studies have been made with Hadoop atop different types of storages (SSD and HDD) and highspeed network interconnects (Infiniband and Ethernet etc). Moon [25] showed significant cost benefit by storing intermediate Hadoop data in SSD, leaving the HDDs to store Hadoop Distributed File System (HDFS [5]) source data. Wu [28] found that Hadoop performance can be increased almost linearly with the increasing fraction of SSDs in the storage system. Ahn [29] identified in a virtual environment overhead of virtualization is minimized with SSDs. Tan [26] analyzed the performance of SSD and

HDD of different type of workloads involving different IO patterns and found better performance in using SSD. Vienne [30] evaluated the performance of Hadoop on different high speed interconnects such as 40GigE RoCE and Inifiniband FDR and found InfiniBand FDR yields the best performance for HPC as well as cloud computing applications. Similarly, Yu [31] found improvedperformance of Hadoop in traditional supercomputers due to high speed networks. From the perspective of a cost effective deployment, Appuswamy [32] studied the scale-out and scale-up performance for different enterprise level Hadoop job and found better perfromance price to performance in scaled up system. On the contrary, Michael [33] reached entirely different conclusion for interactive query. All of the above studies have been performed either with existing benchmarks like HiBench[] or for enterprise level analytics workloads such as log processing etc, thus, unable to address the HPC aspect of Hadoop in terms of efficient hardware provisioning. Furthermore, very limited studies consider the in-memory graph processing frameworks like Giraph, although, graph analysis is a core part of many analytics workloads.

From the above survey, we found a gap in the existing studies in terms of evaluating different distributed cyber infrastructure for current state of the art bigdata analytics software for a real world data intensive scientific workloads. On the other hand, millions of dollars are being invested in programs like NSFcloud with the goal "to provide an experimental platform enabling the academic research community to drive research on a new generation of innovative applications of cloud computing and cloud computing architectures" [46] . Hence, we found it extremely important to find out the limitations in traditional supercomputers in terms of underlying hardware infratructure and present a study addressing how to alleviate those limitations in an efficient yet cost-effective manner.

III. Bigdata Softwares on Traditional Supercomputers

A. Hadoop

Hadoop was originated as the opensource counter part of Google's Map-Reduce [1]. Hadoop has two different components: Hadoop Distributed File System (HDFS) and a mapreduce programming abstarction. HDFS splits huge volume of data into small disjoint sets called blocks (typically of size 64mb to 128mb) and distributes those accross the cluster. A user defined map function is applied to each blocks parallely in order to extract information from each records in the form of key-value pair. These intermidiate key-value pairs are then partitioned on the basis of keys where each key gets a list of values. Finally, a user defined reduce function is applied to the value-list of each key independently and the final output is written to the HDFS.

B. Giraph

Large scale graph analysis is a core part of many supercomputing workload. Apache Giraph is an iterative in-memory grpah processing framework that is implemented on top of Hadoop's map-reduce implementation. It is originated as the open-source counterpart to Google's Pregel [2]. Giraph is inspired by Bulk Synchronous Parallel model [3] where computation proceeds in supersteps. In each superstep all vertices of the graph excutes different istances of the same progam called vetrex-program simultaneosly without ineracting with other verties which is similar to map tasks of Hadoop After each superstep all the vertices send messages to other vertices normally cotaining the output of its vertex-progam-instance. Once all the messages are recieved by the intended vertices, the next supesrtep starts and the pocess iterates until all the vertices vote to halt simultaeously.

C. Hadoop/Giraph-workload and Traditional-Supercomputing hardwares

Earlier studies [42], [44] as well as our experience show that Hadoop and other softwares in its eco system like Giraph can be useful for data-intensive scientific applications. However, the underlying storage, memory as well as the computation model differs severely from other parallel processing frameworks like MPI. In this section we describe the challenges in a traditional supercomputing environment while handling a Hadoop-enabled HPC workload.

1) Network: In a typical Hadoop job, the data movement is minimal early in the job flow when the mappers carfully consider the data locality. Once the mappers completed their tasks, the intermidiate data is shuffled to the reducers which results in a huge data movement across the cluster. On the other hand, Giraph is more network intensive. The computation phase of each Giraph-superstep is followed by a communication phase which sends a huge amount of messages accross all the Giraph workers. Furthermore, in a Giraph job, the number of TCP-connections increases almost exponentially with increase in number of workers. At these points, the data network is a critical path and its performance and latency directly impact the execution time of the entire workflow.

High performance scientific applications running in a supercomputing environment traditionally use an Infiniband interconnect with high performance and low latency. But, Hadoop and Giraph were developed to work atop cheap clusters of commodity hardware based on an Ethernet network. The java based network communication in both Hadoop and Giraph can hardly take the advantage of the Infiniband.

2) Storage: A typical Hadoop job involves a huge amount of disk-io in different phases. First, in the beginning of the map phase, the input data is read from a distributed filesystem parallely by all the mappers. Normally, the HDFS is used for this purpose which is mounted on the Directly Attached Storage (DAS) device(s) in each of the compute nodes. Then, during the shuffle phase a huge amount of data (intermidiate key-value pair) is written by the mappers and subsequently read by the reducers to/from the local file system which is again mounted on the Directly Attached Storage (DAS) device(s) of each compute node. Finally, at the end of the job, the reducers write the final output on the underlying parallel distributed file system. Giaph, on the other hand, is an in-memory framework. It reads/writes the data from the disk only twice. First, in the beginning of the job when it reads the graph data structure from the dfs. And finally, after the completion of the entire computation it writes the final output to the HDFS.

In a traditional supercomputing environment, each node is normally attached with only one HDD. This configuration puts a practical limitation not only on the total storage-space

available for the HDFS, but also in terms of total number of disk-io operations per second (IOPS). Some variations of Hadoop (eg. MyHadoop etc.) are capable to read/write the data from other parallel file system like Lustre or GPFS which are mounted on dadicated io-servers in an HPC environment Although these versions of Hadoop alleviate the problem of total IOPS by using more disks mounted on Luster or GPFS, there is a tradeoff between total IOPS and total network-traffic. Furthermore, distributing the Shuffled data across dadicated io-servers needs complicated partitioning on the parallel file system which is hardly available in a traditional supercoputing environment. In this paper, we consider only HDFS to evaluate the underlying hardwares.

3) Memory: The performance of a Hadoop job can be improved by providing more memory per node in the compute cluster. At the end of the map-phase, each Hadoop-map task spills a huge amount of data onto the DAS. Providing more memory with properly tuned Hadoop-parameters (io.sort.mb and io.sort.factor) reduce the amount of spilling to disk, thus, improve the performance of a Hadoop job significantly. Furthermore, increasing memory in each node improves the cache-effect during the computation which is extremely beneficial for iterative computation in Giraph. Also, more the memory modules per node, more is the number of memory channels, thus increasing the access parallelism of the processors in each node which can also improve the overall performance.

In a traditional Supercomputing environment, normally, 2GB per core is used as a standard configuration which obviously poses a tradeoff between the number of concurrently running mappers and the amount buffers used by each of them before spilling the output into the disk. Furthermore, for a memory-intensive job, like graph analysis with Giraph that loads a huge amount of data in the memory for iterative computation, the low amount of memory available per node in a traditional supercomputer hinders the caching. Hence, results in lower performance.

IV. The Workload

De novo genome assembly refers to the construction of an entire genome sequence from a large amount of short read sequences when no reference genome is available. De Bruijn graph construction and removal of sequencing errors (tips and bubble) from this graph is central to de novo sequencing. Finally, resolving repeated regions followed by a scaffolding phase produces long size scaffolds that represents a region in the actual genome.

We classified de novo sequencing in three different phases like other assemblers. *a*) De Bruijn graph construction *b*) Graph simplification and *c*) Scaffolding. We store short reads in fastq format in hdfs as input to PGA. In the first phase, we use Hadoop in order to build de Bruijn graph from these short reads. Once the graph is constructed we use Giraph in the subsequent phases to analyze the graph in order to construct appreciably long contigs and scaffolds. In this section we provide a brief overview of each stage of the assembler.

A. De Bruijn graph construction

In our assembler we constructed the de Bruijn graph from the fastq short reads using two MapReduce jobs as follows.

1) Fastq-preprocessing: Each short-read (or, simply read) in a fastq file can be considered as a tuple and it consists of four different lines. The first line is a read-id. The second line is the actual read from the sequencing machine. The third line is an additional line containing some biological information. And the fourth line is a quality-score assigned to that read. In thepreprocessing step of our assembler we invoke a mapper-only Hadoop job which filters out only the read-ids and the corresponding reads from the fatq file(s). It then compute the reverse complement of the read. Finally the read and its reverse complement along with the read-id is written in the HDFS.

2) Build-graph: In the map phase, each read is divided into several short fragments of length k known as k-mer. Two subsequent k-mers are emitted as key-value pairs where the first one (key) represents a vertex in the de Bruijn graph and the second one (value) represents the outgoing edge from the key. Each read emits $(l_r - k + 1)$ kmers as keys where l_r is the length of the read. Again, each k-mer is produced twice, once as the key, and once as the value as mentioned before. Similar process is repeated for the reverse complement of all the reads also. Based upon the value of k the Hadoop-mappers write a huge amount of data to the local file system making the job extremely shuffle-intensive. After the mappers complete, the shuffle phase partitions the intermidiate key-value pairs on the basis of key which effectively collects the edges of the graph emitted from the same source k-mer. Finally, the reduce function aggregates the edges (value-list) of each source k-mer and saves the graph structure in HDFS in adjacency list format.

B. Graph Simplification

The graph produced in the graph construction stage works as the input to the graph simplification stage. In this stage PGA invokes a series of Giraph jobs to simplify the graph. Giraph reads the graph from HDFS in an adjacency list format. Each Giraph job consists of three different types of computation, called compression, tip-removal and bubble-removal as described below. The Giraph-master vertex computation keeps a counter on number of supersteps and invoke these three different types of computation based upon that counter.

Compression: The first step that follows after building the graph is compressing the linear chains of nodes in the graph. The non-branching linear paths of vertices are compressed into single vertex without any loss of any information. In one superstep each compressible vertex is tagged as either head or tail with equal probability and send a meassage containing the tag to the immidiate predecessor. In the next superstep the head-vertices are merged with corresponding tail-vertices. The value of the head is updated accordingly. This process continues for i supersteps until there is no compressible vertex remaining in the graph.

Tip removal: Tips are formed because of errors in the end of the short reads. Removing the tips from the de Bruijn Graph is a straight forward process. After compressing the graph, in a single superstep the vertices with no outgoing edge and the value-length less than a threshold (normally set to $2k$) are deleted from the graph.

Bubble removal: Bubbles are introduced in the DBG because of errors in the middle of the short reads. Bubbles

are formed when two paths start and end at the same vertices. After compressing the linear chain, the objective of bubble removal is to group the vertices by the same predecessor and successor in the entire graph and from each group keep only the node which has the highest frequency support. In one superstep every node matching this criteria sends the cumulative frequency to their immidiate successor. In the next superstep successor nodes compute difference in frequency and delete all the nodes with lower frequency. Remember we calculated the frequency during the compression phase.

C. Scaffolding

The first step of scaffolding determines which contigs are linked by matepairs, and their relative orientation and separation. By convention, mated reads have the same name except for their suffix (either 1 or 2). PGA therefore finds all mate-linked contigs using a single MapReduce cycle by emitting from the mapper mate messages consisting of the read name without the suffix as the key, and the contig name, read orientation, and read offset as the value. Next, we developed a graph hop method to find the exact path between the linked nodes

V. Input Data

High throughput next generation DNA sequencing machines like Illumina Genome Analyzer produce huge amount of short read sequences typically in the scale of several GigaBytes to Terabytes. Furthermore, the size of the de Bruijn graph built from these vast amount of short reads may be another magnitude higher than the reads itself making the entire assembly pipe line severely data-intensive.

In this paper, we use the bumble bee genome sequence as a representative data set. The bumble bee genome is available in Genome Assembly Gold-standard Evaluation (GAGE [41]) website in fastq format. The data size is 90GB containing almost 1billion reads. The size of the de Bruijn graph produced by it is 95GB. Table-I shows the details of the entire bumble-bee genome assembly pipeline and the total amount of data read/written in its different stages.

VI. Evaluation Methodology

A. Experimental Testbeds

Table-II shows the description of the experimental testbeds that we use in our study. We use the configuration of LSU supercomputing resource, SuperMikeII as the baseline and compare all the performance result of our private cloud infrastructures, SwatIII and CeresII to this baseline. Each node in any SwatIII variant has the same number of processors and cores as in SuperMikeII. They only vary in terms of the number of disks per node, type of storage media and the amount of memory. The first three variant of SwatIII: SwatIII-Basic, SwatIII-Storage and SwatIII-Memory is used to evaluate the impact of each individual component of a compute cluster, i.e. network-interconnect, storage type and amount of memory per node. SwatIII-basic is similar in every aspect of SuperMikeII except it uses 10-Gbps Ethernet instead of 40-Gbps Infiniband as in SuperMikeII. SwatIII-Storage, as the name suggests, is storage-optimized and use one SSD per node instead of one HDD as in SuperMikeII. Whereas, SwatIII-Memory is both memory and storage optimized, i.e. it uses 1-SSD as well as 256GB memory per node instead of 32GB as in SuperMikeII.

Unlike SuperMikeII or SwatIII-Basic/Storage/Memory which use only one DAS device per workstation, SwatIII-FullScaleup-HDD/SSD and SwatIII-Medium-HDD/SSD use more than one DAS device (Either HDD or SSD as the names suggest) per workstation. They also vary in terms of total amount of memory per node. However, the total amount of storage and memory space is almost same accross all these clusters. We use these clusters to mainly evaluate different types of architectural-balance in terms of raw execution time as well as performance/$. Also, it provides significant insight on how to leverage SSDs in a cloud environment in a cost effective manner. In these configurations we use JBOD (Just a Bunch Of Disks) configuration as per the genral recommendation by [4], Cloudera, Yahoo etc. Use of the JBOD configuration eliminates the limitation on disk-io speed which is constrained by the speed of the slowest disk in case of a RAID (Redundant Array of Independent Disk) configuration. As mentioned in [4], JBOD is found to perform 30% better than RAID-0 in case of HDFS write throughput.

The last one: CeresII is a prototype of MicroBrick-based High-density server for shared nothing paradigm. Unlike SuperMikeII and different variants of SwatIII clusterss, CeresII uses Intel Xeon E3 1220L V2 processor with only 2-cores per server (workstation). It uses 10Gbps Virtual ethernet for the communication across the servers.

B. Hadoop configurations and optimizations

Since our goal is to evaluate the underlying hardwares and the balance among storage, memory and cpu-cores, we avoid any unnecessary change in the source code of Hadoop or Giraph. In order to evaluate the relative merits of different clusters we started with tuning and optimizing different Hadoop parameters to the baseline, that is a traditional supercomputing environment, SuperMikeII. Then, we further modified the parameters with change in the underlying harware infrastructure in SwatIII cluster to optimize the perfromance in each configuration. A brief description of the Hadoop-parameters that we changed are as follows.

Number of Yarn containers: We use a modified version of Hortonworks formula to get number of concurrently running containers to get good performance especially for the clusters where each node is equipped with a single HDD. However, we validated the result of the modified formula by regorous testing by launching different number of containers concurrently.

Amount of memory per container: In each node in any cluster, we kept 10% of the memory available per node both for system use and buffering during the completion of mappers. Rest of the memory is equally divided among the launched containers.

Java Heap Space: We always configure the value of the corresponding Hadoop parameters less the amount of memory allocated per containers which is a general recommendation for any MapReduce job.

Total number of Reducers: We observed the job profile of different workload and fixed the total number of reducers as double the total number of concurrently launched containers

	Nature of the jobs	Input size to the workflow	Final output size of the workflow	Number of jobs	Intermediate data read/written to local file system	Total data read/written to HDFS
Graph Construction	Hadoop	90GB	95GB	2	2TB (shuffle intensive)	136GB
Graph Simplification	Series of Giraph jobs	95GB (71581898 vertices)	24GB (4787619 vertices)	15	-	966GB
Scaffolding	Small Hadoop/Giraph jobs	24GB	640MB	100	????	1121GB

TABLE I: Bumble-bee genome assembly

	Super MikeII	SwatIII-Basic	SwatIII-Storage	SwatIII-Memory	SwatIII-FullScaleup-HDD/SSD	SwatIII-Medium-HDD/SSD	CeresII
Processor	SandyBridge Xeon 64bit Ep series	SandyBridge Xeon 64bit Ep series	SandyBridge Xeon 64bit Ep series	SandyBridge Xeon 64bit Ep series	SandyBridge Xeon 64bit Ep series	SandyBridge Xeon 64bit Ep series	Xeon E3-1220L V2
Processor-speed (GHz)	2.6	2.6	2.6	2.6	2.6	2.6	2.3
#Processor	2	2	2	2	2	2	1
#V-Cores	16	16	16	16	16	16	2
DRAM (GB)	32	32	32	256	256	64	16
DRAM-Speed (MHz)	1600	1600	1600	1600	1600	1600	1600
Storage type	HDD	HDD	SSD	SSD	HDD/SSD	HDD/SSD	SSD
#Disk	1	1	1	1	7	2	1
Disk-Speed	7200RPM	10000 RPM	Random-Read/Write: 100000/90000-IOPS, Sequential-ReadWrite: 540/520 MBps			10000 RPM	
Network	40-Gbps QDR Infiniband (2:1 blocking)	10-Gbps Ethernet	10-Gbps Ethernet	10-Gbps Ethernet	10-Gbps Ethernet	10-Gbps Ethernet	10-Gbps Virtual Ethernet
#Nodes used for Bumble-bee genome assembly	16	16	16	16	3	3	32
#Nodes used for human genome assembly	128	-	-	-	16	16	-

TABLE II: Experimental Tetbeds

always across all clusters which was found to yield good performance.

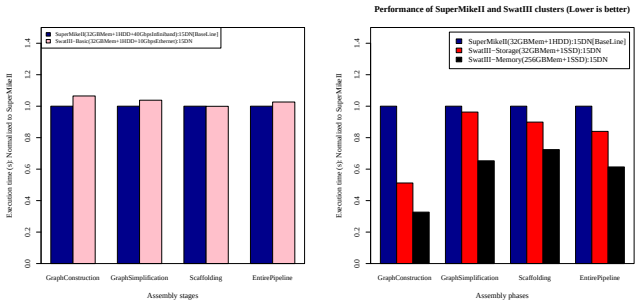
Giraph workers: We changed the numbers of Giraph workers according to the number of Yarn-containers launched simultaneously. Memory per Giraph-worker is fixed similarly to the yarn containers. Other performance parameters: io.sort.mb and io.sort.factor is adjusted according to the cluster-configuration Slow Restart: To segregate the effect of the network during any map-reduce job.

VII. Performance Comparison between SuperMikeII and SwatIII-Basic/Storage/Memory

In this section, we compare the impact of each hardware component: network, storage and memory individually on our benchmark genome assembler. In order to do that, we use 16 nodes both in SuperMikeII and SwatIII. Each node in both the clusters has 16 processing cores. We started with comparing the impact of network between SuperMikeII and SwatIII-Basic. Then, we further optimized the SwatIII cluster incrementally in terms of storage (named as SwatIII-Storage) and memory (named as SwatIII-Memory) one after another and compare the performance of each component in each stage of our assembler.

A. Performance in SuperMikeII

Since each node of SuperMikeII is equipped with only 1-HDD, there is a practical limitation on number of concurrently running yarn containers (hence, mappers and reducers) per



(a) Impact of network (b) Impact of SSD and RAM

Fig. 1: Execution time of different stages of PGA in SuperMikeII and SwatIIIclusters

node. More the number of mappers (or reducers) running simultaneously, more is the parallel disk-io especially during the shuffle phase. With only 1-HDD per node, the performance of Hadoop is advarsely affected because of huge amout of io-wait. As a consequence, even if each node of SuperMikeII has 16 processing cores we observed the best performance for our entire assembly pipeline by running only 8 yarn-containers concurrently in each node, i.e. only half of the number of cores per node.

B. *Effect of Network*

Figure-1a compares the impact of network interconnect on both separately on each stage of the entire genome assembly pipeline while assembling a 90GB bumble bee genome. The execution time is normalized to the SuperMikeII-baseline. That is, the execution time on SuperMikeII for different stages of the asembler always have the value 1. SuperMikeII uses a 40-Gbps QDR Infiniband whereas SwatIII-basic uses 10-Gbps ethernet. Since the java-based APIs of Hadoop is not optimized to take the advantage of Infiniband, in our assembly pipeline we did not observe any visible performance (less than 2%) difference between these two different types of network interconnect.

C. *Effect of SSD*

Figure-1b shows the execution time of our assembler in SuperMikeII and SwatIII-storage which uses 1-SSD per node unlike 1-HDD per node as in SuperMikeII. The execution time is again normalized to the SuperMikeII-baseline. The second columns in each stage of the assembler in Figure-1 shows the impact of using SSD in that stage of the assembly. Graph construction being a shuffle-intensive Hadoop job, got maximum benefit from SSD. We observed almost 60% improvement in this phase of the assembler. Graph simplification consists of a series of in-memory Giraph jobs. Hence, in this stage, we did not observe much impact (less than 5%) of SSD. Scaffolding stage, being a mix of small Hadoop and Giraph job the corresponding gain is almost 10%.

Figure-2 compares the CPU-utilization and IO-wait characteristics for 1-HDD and 1-SSD per node. In the graph construction stage, we started the reducers after all the mappers finished. Hence, the first three peaks in the left side of figure-2a and 2d corresponds to the CPU utilization of three mapper-waves. In case of HDD, most of the io-wait is found to occur in two places. First, at the end of each mapper-wave when many mappers write onto the DAS in parallel. Second, when the shuffled data is copied to the reducers. In the graph simplification stage, we observed fewer io-wait mainly when Giraph reads/writes a large graph from/to HDFS. Scaffolding being a series of small Hadoop and Giraph job suffers from least io-wait. As shown in figure-2d,2e and 2f io-wait is reduced by using solid state drive (SSD) instead of HDD especially in case of shuffle-intensive Hadoop job.

Figure-3 compares the read and write ios per second to the local disk (of one datanode) for different stages of the same assembly using HDD and SSD. We observed almost 7 to 8 times improvement in the peak IOPS in case of SSD than HDD in the shuffle-intensive graph construction phase that writes huge amount of data to the local file system. Figure-4 compares the total HDFS-bytes read/written per second accross the cluster using HDD and SSD. There is almost 2-3 times improvement in the peak HDFS read/write per second for SSD in any of the phase of the assembler.

In a traditional supercomputing environment each compute node is typically provided with only one local hard disk drive (HDD), thus provides fewer number of io-operations per second (IOPS) which makes a Hadoop job severely io-bound. The problem is more severe in case of a shuffle-intensive Hadoop job which involves huge amount parallel ios to the local file system.

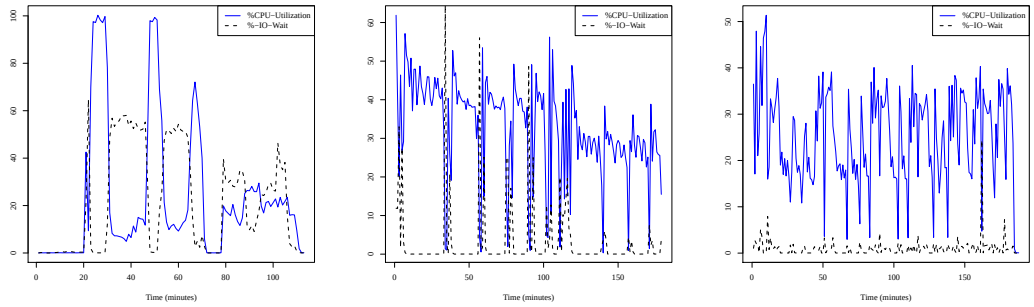
Figure-2 compares the CPU-utilization and IO-wait characteristics for both 1-HDD and 1-SSD. We observed, Figiowait(a) shows the huge amount of io-wait that we observed in the graph construction phase of our benchmark-assembler while assembling the bumble bee genome using 16 nodes each with only one HDD. We observed the maximum io-wait at the end of each mapper-wave when a huge amount of data is written to the local file system.

D. *Effect of DRAM*

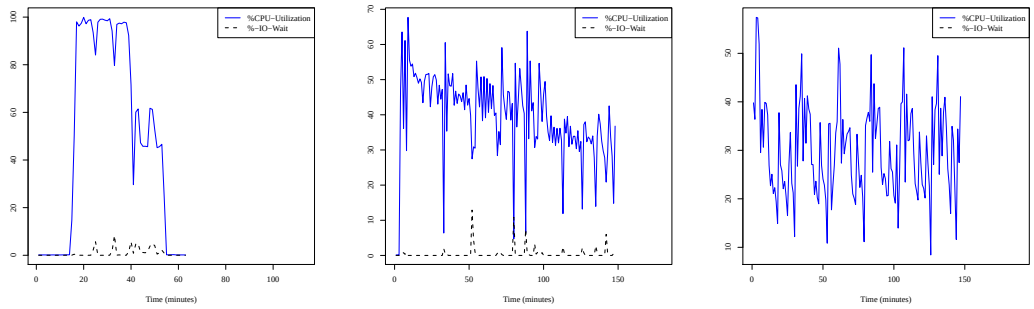
The third columns of Figure-1b shows the impact of memory in different stages of our assemblers in SwatIII-Memory normalized to the SuperMikeII-baseline. We observed almost 20% improvement in the initial graph-construction phase from SwatIII-Storage and almost 70% improvement to the baseline. Both SwatIII-Storage and SwatIII-Memory use SSD as their underlying storage. Due to increase in the memory size, there is fewer amount of data spilling to the disk at the end of the map phase. In the Giraph phase, the corresponding improvement is almost 40%. The computation in Giraph proceeds in iterative supersteps. Given enough memory, a huge amount of data is kept in cache and is fetched upon requirement during the next compute-superstep.

VIII. Comparing Different Architectural-Balance

In this section we compare the performance of different cluster architecture in terms of raw execution time as well as performance per dollar. Figure-5a shows the relative marits of different cluster architecture in terms of raw execution time. Since most of the time the selection of the clusters are driven



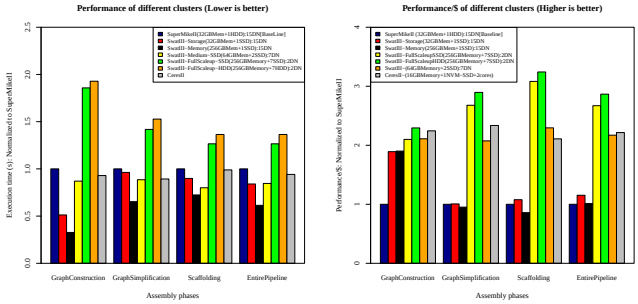
(a) CPU-Util in Graph Construction in SwatIII-Basic (b) CPU-Util in Graph Simplification in SwatIII-Basic (c) CPU-Util in Scaffolding in SwatIII-Basic



(d) CPU-Util in Graph Construction in SwatIII-Storage (e) CPU-Util in Graph Simplification in SwatIII-Storage (f) CPU-Util in Scaffolding in SwatIII-Storage

Fig. 2: CPU-Utilization and IO-Wait characteristics in SwatIII-Basic (1-HDD/node) and SwatIII-Storage(1-SSD/node)

by the sheer volume of data that needs some minimum storage and memory space to be analyzed, in our study, we did not compromise the total storage or memory space. All the clusters that we evaluate in this section has almost the same amount of total memory and storage space except SwatIII-Memory where the amount of memory is significantly higher than the others. The observations are as follows: 1) The SwatIII-Memory, i.e.



(a) Assembly time comparison (b) Performance/$ comparison

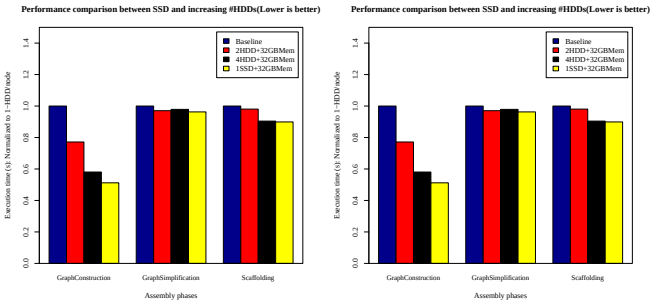
Fig. 5: Compare different type of cluster architeture for Bumble bee genome assembly pipeline

the 16-nodes cluster with 256GB memory and one SSD per node performs the best in terms of raw execution time for any type of workload due to high resource availability. 2) Given the same amount of storage and memory space and same type of storage, Hadoop performs almost linearly with increase in number of nodes because of increase in number of cores, as shown in SwatIII-Storage (16nodes), SwatIII-Medium-SSD(8nodes) and SwatIII-FullScaleup-SSD (2nodes). 3) Number of cores plays a critical role in case of Giraph. We observed the optimum performance in graph simplification stage in SwatIII-Medium (8-nodes) cluster. 4) Although the use of SSD is beneficial for Hadoop when there is only one disk per node (as shown in SuperMikeII and SwatIII-Storage), in a full scaled-up environment with multiple disks per node, Hadoop shows similar performance with both HDD and SSD as shown in SwatIII-FullScaleup-SSD and SwatIII-FullScaleup-HDD. 5) SSD yields better scalability than HDD when more nodes are added to the cluster. It can be observed by comparing the execution time of graph construction stage in 2, 7 and 15 datanode separately for SSD and HDD variant of Swat-III.

A. *Scaledup cluster and SSD*

In order to quantify the benefit of SSD, we started with evaluating the performance of a single SSD per node in the SwatIII-Storage cluster. We replaced the single SSD used in each node of SwatIII-Storage with increasing number of HDDs and assembled the 90GB bumble-bee genome each time using 16nodes until the execution time is similar to that

of the single SSD case. Figure-6a shows the execution time of different number of HDDs per node again normalized to the baseline. We observed almost a linear trend in performance by increasing the number of HDDs per node in the cluster. At the same time, 4-HDDs per node shows similar performance (only 5% variation) with a single-SSD in the graph-construction phase. Hence, at this point, we expected a similar performance for both HDDs and SSDs with more than 4 DAS per node in any compute cluster with the same number of cores per node. We generalize the observation



(a) Performance trend using 1, 2 and 4 HDD(s) and 1-SSD per node using a 16-node cluster

(b) Performance trend for SSD and HDD using 1, 2, 7 disks per node in 16, 8 and 2-nodes cluster

Fig. 6: Comparing HDDs and SSDs

and propose the following rule for a shuffle-intensive Hadoop job. If, $(TotalShuffledData\ /\ NumNodesInCluster)/SingleDiskCapacity < ThresholdPoint$ then, SSD is benefitial in terms of performance, where the $ThresholdPoint$ is determined by number of cores and the processor family. In our case the $ThresholdPoint$ is 4

B. CeresII: Scaledout-in-a-box and SSD

In the previous sections we observed that SSD shows huge performance benefit in a scaled out cluster setup where each node has fewer number of DAS. We also observed that the performance gap between HDD and SSD decreases with increase in number of DAS per node. Finally, depending upon the number of cores (and processor family) per node, beyond a certain threshold point HDD and SSD starts perform similarly. However, due to less number of cores in fewer nodes, the overall performance may drop significantly. On the other hand, use of more scaledup servers has an immidiate impact on performance to price.

In this section, we evaluate another cluster architecture called CeresII which uses 2 physical cores per node, 1 NVM SSD and 16GB memory per node. In order to assemble the 95GB bumblebee genome we use 33 nodes of this cluster. The last columns of different stages of the assembly in Figure-5 shows the execution time of CeresII. As it can be seen, CeresII with only half the number of cores than the SupermikeII-baseline performs better in every stage of the assembly pipeline.

IX. Price to Performance

Table-price show the cost of each hardware component and the entire workstation used in different experiments in this paper. It is worthy to mention here, We do not assume that a single scaled up server with one disk can accomodate the entire data. The total amount of data should be held in its entirity in the cluster for both scaled-up and scaled-out cases. Hence, we did not compromise with the total disk-space or memory-space required in case of scaled up and scaled out. Rather, we compare the performance to price from the view point of a proper architectural balance among number of cores, number of disks and amount of memory. Since, we did not find any impact of network on the oerformance of our assembly pipeline we exclude the cost of network (infiniband and ethernet) in our comparison. It is obvious, that Infiniband with higher cost will yield lower performance to price as we did not find much performance difference between these two network interconnect. Figure-5b shows the performance to price comparison among all the clusters.

X. Conclusion

Acknowledgment

The authors would like to thank...

References

[1] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51, no. 1 (2008): 107-113.

[2] Malewicz, Grzegorz, Matthew H. Austern, Aart JC Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. "Pregel: a system for large-scale graph processing." In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp. 135-146. ACM, 2010.

[3] Cheatham, Thomas, Amr Fahmy, Dan Stefanescu, and Leslie Valiant. "Bulk synchronous parallel computinga paradigm for transportable software." In Tools and Environments for Parallel and Distributed Systems, pp. 61-76. Springer US, 1996.

[4] White, Tom. Hadoop: the definitive guide: the definitive guide. " O'Reilly Media, Inc.", 2009.

[5] Borthakur, Dhruba. "The hadoop distributed file system: Architecture and design." Hadoop Project Website 11, no. 2007 (2007): 21.

[6] George, Lars. HBase: the definitive guide. " O'Reilly Media, Inc.", 2011.

[7] Thusoo, Ashish, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. "Hive: a warehousing solution over a map-reduce framework." Proceedings of the VLDB Endowment 2, no. 2 (2009): 1626-1629.

[8] Wanderman-Milne, Skye, and Nong Li. "Runtime Code Generation in Cloudera Impala." IEEE Data Eng. Bull. 37, no. 1 (2014): 31-37.

[9] Chen, Rishan, Xuetian Weng, Bingsheng He, Mao Yang, Byron Choi, and Xiaoming Li. "On the efficiency and programmability of large graph processing in the cloud." Microsoft Research TechReport (2010).

[10] Kang, U., Charalampos E. Tsourakakis, and Christos Faloutsos. "Pegasus: A peta-scale graph mining system implementation and observations." In Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on, pp. 229-238. IEEE, 2009.

[11] Kang, U., Hanghang Tong, Jimeng Sun, Ching-Yung Lin, and Christos Faloutsos. "Gbase: a scalable and general graph management system." In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 1091-1099. ACM, 2011.

[12] Zaharia, Matei, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, pp. 2-2. USENIX Association, 2012.

	Super MikeII	SwatIII-Basic	SwatIII-Storage	SwatIII-Memory	SwatIII-FullScaleup-SSD	SwatIII-FullScaleup-HDD	CeresII
Processor ($)	3040	3040	3040	3040	3040	3040	389
Memory ($)	279	279	279	279	279	279	232
Disk ($)	67	67	177	177	177	177	140
Total-Cost/ Work-Station ($)	3386	3386	3496	5449	6511	5741	761
Nodes used for Bumble-bee genome assembly	15	15	15	15	2	2	32
Cost of the cluster ($)	50790	50790	52240	81735	13022	11482	24352

TABLE III: Cost of Different Clusters (the price of different components are collected from amazon.com)

[13] Zaharia, Matei, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. "Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters." In Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing, pp. 10-10. USENIX Association, 2012.

[14] Gonzalez, Joseph E., Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. "Graphx: Graph processing in a distributed dataflow framework." In Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI). 2014.

[15] Xin, Reynold S., Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. "Shark: SQL and rich analytics at scale." In Proceedings of the 2013 ACM SIGMOD International Conference on Management of data, pp. 13-24. ACM, 2013.

[16] Carlson, Josiah L. Redis in Action. Manning Publications Co., 2013.

[17] Power, Russell, and Jinyang Li. "Piccolo: Building Fast, Distributed Programs with Partitioned Tables." In OSDI, vol. 10, pp. 1-14. 2010.

[18] Avery, Ching. "Giraph: Large-scale graph processing infrastructure on hadoop." Proceedings of the Hadoop Summit. Santa Clara (2011).

[19] Tian, Yuanyuan, Andrey Balmin, Severin Andreas Corsten, Shirish Tatikonda, and John McPherson. "From" think like a vertex" to" think like a graph." Proceedings of the VLDB Endowment 7, no. 3 (2013): 193-204.

[20] Low, Yucheng, Joseph E. Gonzalez, Aapo Kyrola, Danny Bickson, Carlos E. Guestrin, and Joseph Hellerstein. "Graphlab: A new framework for parallel machine learning." arXiv preprint arXiv:1408.2041 (2014).

[21] Xin, Reynold S., Joseph E. Gonzalez, Michael J. Franklin, and Ion Stoica. "Graphx: A resilient distributed graph system on spark." In First International Workshop on Graph Data Management Experiences and Systems, p. 2. ACM, 2013.

[22] Roy, Amitabha, Ivo Mihailovic, and Willy Zwaenepoel. "X-stream: Edge-centric graph processing using streaming partitions." In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, pp. 472-488. ACM, 2013.

[23] Kyrola, Aapo, Guy E. Blelloch, and Carlos Guestrin. "GraphChi: Large-Scale Graph Computation on Just a PC." In OSDI, vol. 12, pp. 31-46. 2012.

[24] Krishnan, Sriram, Mahidhar Tatineni, and Chaitanya Baru. "myHadoop-Hadoop-on-Demand on Traditional HPC Resources." San Diego Supercomputer Center Technical Report TR-2011-2, University of California, San Diego (2011).

[25] Moon, Sangwhan, Jaehwan Lee, and Yang Suk Kee. "Introducing SSDs to the Hadoop MapReduce Framework." In Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on, pp. 272-279. IEEE, 2014.

[26] Tan, Wei, Liana Fong, and Yanbin Liu. "Effectiveness Assessment of Solid-State Drive Used in Big Data Services." In Web Services (ICWS), 2014 IEEE International Conference on, pp. 393-400. IEEE, 2014.

[27] Kang, Yangwook, Yang-suk Kee, Ethan L. Miller, and Chanik Park. "Enabling cost-effective data processing with smart ssd." In Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on, pp. 1-12. IEEE, 2013.

[28] Wu, Dan, Wenhai Luo, Wenyan Xie, Xiaoheng Ji, Jian He, and Di Wu. "Understanding the Impacts of Solid-State Storage on the Hadoop Performance." In Advanced Cloud and Big Data (CBD), 2013 International Conference on, pp. 125-130. IEEE, 2013.

[29] Ahn, Sungyong, Sangkyu Park, Jae-Ki Hong, and Wooseok Chang. "Performance Implications of SSDs in Virtualized Hadoop Clusters." In Big Data (BigData Congress), 2014 IEEE International Congress on, pp. 586-593. IEEE, 2014.

[30] Vienne, Jerome, Jitong Chen, Md Wasi-Ur-Rahman, Nusrat S. Islam, Hari Subramoni, and Dhabaleswar K. Panda. "Performance analysis and evaluation of infiniband fdr and 40gige roce on hpc and cloud computing systems." In High-Performance Interconnects (HOTI), 2012 IEEE 20th Annual Symposium on, pp. 48-55. IEEE, 2012.

[31] Yu, Jie, Guangming Liu, Wei Hu, Wenrui Dong, and Weiwei Zhang. "Mechanisms of Optimizing MapReduce Framework on High Performance Computer." In High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on, pp. 708-713. IEEE, 2013.

[32] Appuswamy, Raja, Christos Gkantsidis, Dushyanth Narayanan, Orion Hodson, and Antony Rowstron. "Scale-up vs Scale-out for Hadoop: Time to rethink?." In Proceedings of the 4th annual Symposium on Cloud Computing, p. 20. ACM, 2013.

[33] Michael, Maged, Jose E. Moreira, Doron Shiloach, and Robert W. Wisniewski. "Scale-up x scale-out: A case study using nutch/lucene." In Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, pp. 1-8. IEEE, 2007.

[34] Chen, Yanpei, Sara Alspaugh, and Randy Katz. "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads." Proceedings of the VLDB Endowment 5, no. 12 (2012): 1802-1813.

[35] Huang, Shengsheng, Jie Huang, Yan Liu, Lan Yi, and Jinquan Dai. "Hibench: A representative and comprehensive hadoop benchmark suite." In Proc. ICDE Workshops. 2010.

[36] Murphy, Richard C., Kyle B. Wheeler, Brian W. Barrett, and James A. Ang. "Introducing the graph 500." Cray Users Group (CUG) (2010).

[37] Marathe, Aniruddha, Rachel Harris, David K. Lowenthal, Bronis R. de Supinski, Barry Rountree, Martin Schulz, and Xin 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, pp. 239-250. ACM, 2013.

[38] Pevzner, Pavel A., Haixu Tang, and Michael S. Waterman. "An Eulerian path approach to DNA fragment assembly." Proceedings of the National Academy of Sciences 98, no. 17 (2001): 9748-9753.

[39] Medvedev, Paul, Eric Scott, Boyko Kakaradov, and Pavel Pevzner. "Error correction of high-throughput sequencing datasets with non-uniform coverage." Bioinformatics 27, no. 13 (2011): i137-i141.

[40] Yang, Xiao, Sriram P. Chockalingam, and Srinivas Aluru. "A survey of error-correction methods for next-generation sequencing." Briefings in bioinformatics 14, no. 1 (2013): 56-66.

[41] Salzberg, Steven L., Adam M. Phillippy, Aleksey Zimin, Daniela Puiu, Tanja Magoc, Sergey Koren, Todd J. Treangen et al. "GAGE: A critical evaluation of genome assemblies and assembly algorithms." Genome research 22, no. 3 (2012): 557-567.

[42] Fadika, Zacharia, Madhusudhan Govindaraju, Richard Canon, and Lavanya Ramakrishnan. "Evaluating hadoop for data-intensive scientific

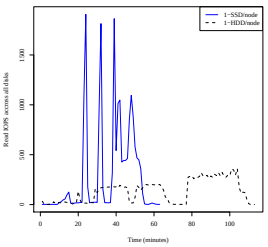
operations." In Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, pp. 67-74. IEEE, 2012.

[43] Jha, Shantenu, Judy Qiu, Andre Luckow, Pradeep Mantha, and Geoffrey C. Fox. "A tale of two data-intensive paradigms: Applications, abstractions, and architectures." In Big Data (BigData Congress), 2014 IEEE International Congress on, pp. 645-652. IEEE, 2014.

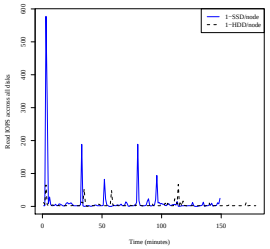
[44] Matsunaga, Andra, Maurcio Tsugawa, and Jos Fortes. "Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications." In eScience, 2008. eScience'08. IEEE Fourth International Conference on, pp. 222-229. IEEE, 2008.

[45] Guo, Yong, Marcin Biczak, Ana Lucia Varbanescu, Alexandru Iosup, Claudio Martella, and Theodore L. Willke. "How well do graph-processing platforms perform? an empirical performance evaluation and analysis." In Parallel and Distributed Processing Symposium, 2014 IEEE 28th International, pp. 395-404. IEEE, 2014.

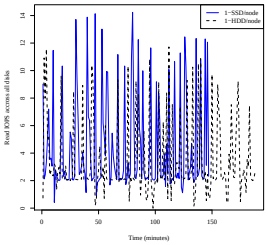
[46] https://www.chameleoncloud.org/nsf-cloud-workshop/.



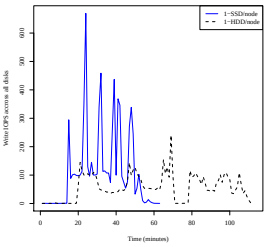
(a) Read-Iops in Graph Construction in SwatIII-Basic and SwatIII-Storage



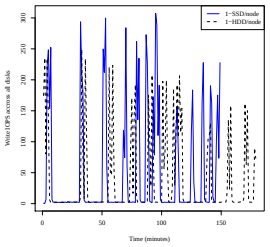
(b) Read-Iops in Graph Simplification in SwatIII-Basic and SwatIII-Storage



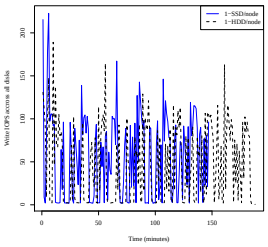
(c) Read-Iops in Scaffolding in SwatIII-Basic and SwatIII-Storage



(d) Write-Iops in Graph Construction in SwatIII-Basic and SwatIII-Storage

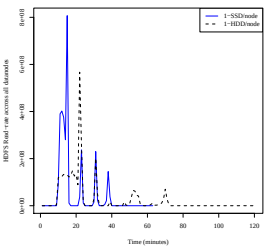


(e) Write-Iops in Graph Simplification in SwatIII-Basic and SwatIII-Storage

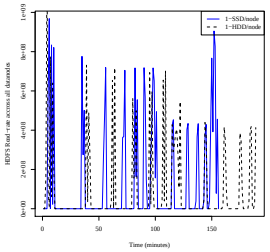


(f) Write-Iops in Scaffolding in SwatIII-Basic and SwatIII-Storage

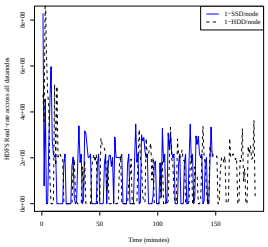
Fig. 3: Comparison of Total IOPS in one host in SwatIII-Basic (1-HDD/node) and SwatIII-Storage (1-SSD/node)



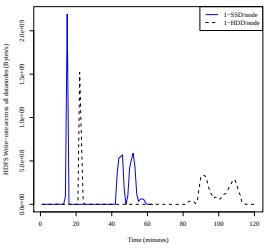
(a) HDFS-Read-Rate in Graph Construction in SwatIII-Basic and SwatIII-Storage



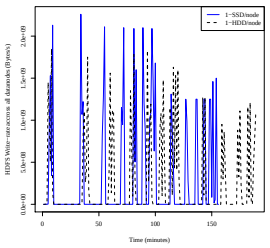
(b) HDFS-Read-Rate in Graph Simplification in SwatIII-Basic and SwatIII-Storage



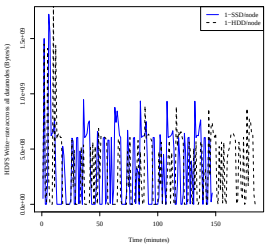
(c) HDFS-Read-Rate in Scaffolding phase in SwatIII-Basic and SwatIII-Storage



(d) HDFS-Write-Rate in Graph Construction in SwatIII-Basic and SwatIII-Storage



(e) HDFS-Write-Rate in Graph Simplification in SwatIII-Basic and SwatIII-Storage



(f) HDFS-Write-Rate in Scaffolding phase in SwatIII-Basic and SwatIII-Storage

Fig. 4: Comparison of HDFS read/write rate across all datanode SwatIII-Basic (1-HDD/node) and SwatIII-Storage (1-SSD/node)