Evaluating Different Distributed-Cyber-Infrastructure for Data and Compute Intensive Scientific Application

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

Jaeki Hong, Wooseok Chang
Samsung Electronics Co., Ltd.
Giheung-gu
Yongin-si, Gyeonggi-do, 446711
Email: {jaeki.hong, wooseok_chang} @samsung.com

***Abstract*—The enormous growth in the amount of data that the various experimental-devices generate is rapidly changing the computational model. Recently, data and compute intensive computation frameworks, such as Hadoop and Giraph, have emerged as bigdata analytic softwares. In particular, scientists are increasingly using these softwares to efficiently handle these big-data, deviating from traditional MPI or grid-based technologies . However, there is limited understanding that how the different types of hardware-architectures impact the performance of these bigdata analytics softwares when applied to a real world data and compute intensive scientific workload.**

In this study, we evaluated the performance of the bigdata analytics softwares over different hardware architectures including HPC clusters (i.e., SuperMikeII) and two different types of private cloud infrastructures (e.g., SwatIII based on regular data center architecture and CeresII based on new microbrick architecture) using our own benchmark software package (i.e., Parallel Genome Assembler (PGA) developed atop Hadoop and Giraph) serving as a very good real world example of data as well as compute intensive workload.

Comparing with the individual impact of different hardware components (e.g. network, storage and memory) over different clusters, we observed 70% improvement in the Hadoop-workload and almost 35% improvement in the Giraph-workload in the SwatIII cluster over SuperMikeII by using SSD (thus, increasing the disk-IO rate) and scaling it up in terms of memory (which increases the caching). Then, we provide significant insight on efficient and cost-effective organization of these hardware components in the entire compute clusters. In this part, the CeresII prototype-cluster is found to yield same level of performance as in SuperMikeII while yielding more than 2-times improvement in performance per dollar in the entire benchmark test.

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 have 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 applications. As a result, providing efficient and cost-effective hardwares became more challenging, however, openning new opportunities for the hardware-manufacturers. Furthermore, in the last few years, an increasing 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-effective, 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[1] where several academic organizations and manufacturing companies collaborated to address the challenges involved in developing these novel distributed cyber infrastructures.

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 hardwares 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

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

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 largescale graph analysis, thus, serving as a very good example of both data- as well as compute-intensive workload.

Specifically, in this paper, we compre 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 as follows. 1) A traditional HPC cluster, called SuperMikeII, located in LSU, USA. 2) A regular data center architecture, called SwatIII, located in Samsung, Korea and 3) A new microbrick based prototype architecture, called CeresII, also located in Samsung, Korea. Our performance analysis is divided into two parts as follows:

1) In the first part, comparing the individual impact of different hardware component over different clusters, we observe almost 70% improvement in the Hadoop-based graph construction stage and 35% improvement in the Giraph based graph-simplification stage in the SwatIII cluster over SuperMikeII by using SSD and scaling it up in terms of memory. SSD increases the disk-io rate, thus reducing the io-wait. Whereas, more memory increases the caching effect.
2) Then, in the second part we provide significant insight on efficient and cost-effective organization of different hardware components. In this part we modified the underlying hardware organization of SwatIII (the regular datacenter architecture) in many different ways to better understand the impact of different architectural balance. In this part, we observe that after a certain threshold on the number of disks per node, SSD and HDD yields similar performance because of similar amount of disk-io rate. Where the threshold is determined by the number of cores. We also observed that the new microbrick based prototype architecture, CeresII is found to yield almost similar performance as SuperMikeII while yielding almost 2-times improvement in performance/$.

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 [1] [2] as well as our experience shows that state-of-the-art bigdata analytics software (e.g. Hadoop etc.) can be useful for HPC workloads involving huge amount of bigdata. Jha [2] in his study nicely showed the convergence between the two paradigms: the traditional HPC-softwares and the Apache Software Stack for bigdata analytics. As a consequence, a growing number of codes in several scientific areas such as bioinformatics, geoscience are currently being written using Hadoop, Giraph etc. [3]. Many of the traditional supercomputers also started using myHadoop [3] to provide the scientists an easy interface to configure Hadoop on-demand. Despite of the growing popularity of using Hadoop and other softwares in its rich ecosystem for scientific-computing, there are very limited prior works that evaluated different distributed cyber infrastructures for these softwares when applied for data-intensive scientific workload. In this section we provide the related works for our study.

Impact of individual hardware component on Hadoop-workload: There are several performance analysis study on using different types of hardwares to accelerate the Hadoop job using the existing benchmark workloads. Vienne [4] 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 [5] found improvedperformance of Hadoop in traditional supercomputers due to high speed networks.

Kang [6] compared the execution time of sort, join, WordCount, Bayesian, and DFSIO workloads using SSD and HDD and obtained better performance using SSD. Wu [7] found that Hadoop performance can be increased almost linearly with the increasing fraction of SSDs in the storage system. They used the tera-sort benchmark for their study. Additionally, they also showed that in an SSD-dominant cluster Hadoop-performance is almost insensitive of different Hadoop performance parameter like block-size and buffer-size. Moon [8], showed a significant cost benefit by storing the intermediate Hadoop data in SSD, leaving the HDDs to store Hadoop Distributed File System (HDFS [9]) source data. They also used the terasort benchmark in their study. Similar result can be found in the study by Li [10] and Krish [11] where SSDs are used to serve temporary data to reduce disk contention and used HDDs to store the HDFS data. They all reached the same conclusion as [8]. Tan [12] also reached the similar conclusion for two other workloads including a Hive-workload and an HBase-workload.

All of the above studies have been performed either with existing benchmarks like HiBench [13] or for enterprise level analytics workloads, thus, unable to address the HPC aspect of Hadoop.

Furthermore, very limited studies consider the in-memory graph processing frameworks like Giraph, although, graph analysis is a core part of many analytics workloads.

Impact of overall archtecticture on Hadoop Workload Michael [14] investigated the performance characteristics of the scaled-out and scaled-up architecture for interactive queries and found better performance using a scaledout cluster. On the other hand, Appuswamy [15] reached entirely different conclusion in their study. They observed a single scaled-up server to perform better than a 8-nodes scaled-out cluster for eleven different enterprise-level Hadoop workloads including log-processing, sorting, Mahout-machine-learning etc. Our study

is significantly different in the following aspects. 1)Existing works mostly focus on enterprise level Hadoop job. Our Hadoop enabled genome assembly workload is significantly different. Additionally, we cover a Giraph workload. 2) Unlike the existing works, we consider the entire workflow of a genome assembly workload instead of chosing a single job, thus working closer to the real world. 3) Existing works are limited in terms of data size. For example, the data size chosen in [15] can be accomodated in a single scaled-up server. On the contrary, we did not put such a restriction. Consequently, we evaluate the performance of scaled-up cluster with fewer nodes instead of evaluating only one scaled-up server. All of our clusters provide almost same amount of storage and memory. Our analysis is more generic and realistic in a sense, that most of the time the choice of the cluster-size is driven by the data size rather than the performance.

III. BIGDATA SOFTWARES ON TRADITIONAL SUPERCOMPUTERS

A. Hadoop

Hadoop was originated as the opensource counter part of Google's Map-Reduce [16]. 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 [17]. Giraph is inspired by Bulk Synchronous Parallel model [18] 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 [1], [19] 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 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/to other parallel file system like Lustre or GPFS which are mounted on dadicated io-servers in an HPC environment. Although these versions of Hadoop can be well optimized to take the advantage of huge amount of disk-IOPS availale through the dadicated io-servers (that are used to mount the Luster or GPFS), the performance can be severely constrained by the available network bandwidth which is shared among many users, consequently generating huge 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 map task spills a huge amount of data onto the DAS. Providing more memory with properly tuned buffer-size (io.sort.mb and io.sort.factor) reduce the amount of spilling to the disk, thus, improve the performance of a Hadoop job significantly. Furthermore, increasing memory in each node improves the caching 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 of 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 per node 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. Hadoop-based 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. Giraph-based 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: Mix of Hadoop and Giraph job

Scaffolding consists of mainly two different types of jobs as follows:

Mate Bundling: 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). Contrail 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. The reduce function scans these mate messages and saves contig link messages that contain the id of the two contigs that are linked and their expected separation and relative orientation. A second MapReduce stage emits the contig link messages and the assembly graph in the mapper, and then bundles together link messages between the same pair of contigs in the reducer (Figure 46, left). Mate-pairs between repetitive contigs are discarded, using a threshold on contig depth of coverage to filter repetitive contigs.

Bundle Resolution: Once the mates are bundled, Contrail searches the graph for paths of contigs consistent with bundles supported by multiple mate pairs (default 5). If there are multiple such bundles extending from the forward or reverse of a contig, then only the bundle to the nearest contig is considered. More distant connections are considered in subsequent rounds of scaffolding. A path is consistent if the separation between contigs implied by the path of overlapping contigs is within the expected distance recorded in the bundle, and their relative orientation matches the relative orientation implied by the mate pairs. If a unique path is found to be consistent with the bundle, it merges the contigs along that path into a single contig (Figure 46, right). For this, Contrail uses a variation of breath-first frontier search from all unique nodes with bundles. In the first MapReduce cycle, all paths of length 1 are explored, using message passing between the unique nodes and their immediate neighbors. In the second cycle, all paths of length 2 are explored using message passing from the 1-hop neighbors in the first cycle. The process repeats for n cycles (default 20), iteratively exploring more distant neighbors. Each hop adds at least one extra base to the candidate path, but will usually extend the path by a much larger stride, depending on the contig size. In each cycle, paths longer (in total base-pairs) than the expected distance to the mate-linked contig are pruned from further consideration. Paths ending with the correct separation and orientation at the matelinked contig are stored. If after n cycles, there is only a single path consistent with the bundled mate pairs, the path of contigs connecting those contigs are resolved into individual larger contigs using a variation of the repeat resolution method described above. In short, repetitive contigs along the path are split into multiple copies, depending on how many paths contain them, and then the linear path compression routine merges the path of now non-branching contigs into a single contig. The scaffolding process then repeats bundling and merging nodes until no more merges occur.

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.

	Job Type	Input	Final output	# jobs	Shuffled data	HDFS Data
Graph Construction	Hadoop	90GB	95GB	2	2TB	136GB
Graph Simplification	Series of Giraph jobs	95GB (71581898 vertices)	24GB (4787619 vertices)	15	-	966GB
Scaffolding	Small Hadoop/ Giraph	24GB	640MB	100	600GB	1121GB

TABLE I: Bumble-bee genome assembly

	Job Type	Input	Final output	# jobs	Shuffled data	HDFS Data
Graph Construction	Hadoop	452GB	3TB	2	9.9TB	3.2TB
Graph Simplification	Series of Giraph jobs	3.2TB (1483246722 vertices)	24GB (16619505 vertices)	15	-	4.1TB

TABLE II: Human genome assembly

In this paper, we use two genome dataset, 1) a moderate size bumble bee genome data (90GB) and 2) a large scale human genome data (452GB). The corresponding graph size is 95GB and 3.2TB based upon the number of unique k-mers in the data set The bumble bee genome is available in Genome Assembly Gold-standard Evaluation (GAGE [20]) website[2] in fastq format. The Human genome is available in NCBI website with accession number SRA0016239. Table-I and II shows the details of the data size in the assembly pipeline for both the genomes. It is to be noted that, in SuperMikeII we could not run the scaffolding stage for the human genome data set because of lower number of maximum allowed open file descriptor ($ulimit - n$). Hence, we did not consider it in our comparison. The corresponding parameter was set to 1024 in SuperMikeII whereas it is quite common to make to set it to some higher value (eg. 64000).

VI. Evaluation Methodology

A. Experimental Testbeds

Table-III shows the overview 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 of the SwatIII variant has the same number of processors and cores as in SuperMikeII. In particular, each SuperMikeII- and SwatIII-node uses two Intel SandyBridge Xeon 64bit Ep series processor. Each of the processor has 8-cores, thus yielding 16 physical cores per node. 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

[2]http://gage.cbcb.umd.edu/

	Super MikeII	SwatIII-Basic	SwatIII-Storage	SwatIII-Memory	SwatIII-FullScaleup-HDD/SSD	SwatIII-Medium-HDD/SSD	CeresII
#Processor/workstation	2	2	2	2	2	2	1
#Physical-Cores/workstation	16	16	16	16	16	16	2
DRAM(GB)/workstation	32	32	32	256	256	64	16
#Disks(500GB each)/workstation	1-HDD	1-HDD	1-SSD	1-SSD	7-HDD/SSD	2-HDD/SSD	1-SSD
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
#DataNodes used for bubmble bee genome (90GB)	15	15	15	15	4	2	31
#DataNodes used for human genome (452GB)	128	-	-	-	16	-	-

TABLE III: Experimental Tetbeds

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. On the other hand, 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 worstation. 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 hardware-organization and architectural-balance in terms of raw execution time as well as performance/$. In either of SwatIII-FullScaleup and SwatIII-Medium, we use JBOD (Just a Bunch Of Disks) configuration as per the genral recommendation by [21], 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 [21], JBOD is found to perform 30% better than RAID-0 in case of HDFS write throughput.

The last one: CeresII is an improvement over CeresI [22] and is in prototype phase. The architecture of CeresII is based on Samsung-MicroBricks. A single MicroBricks chassis consists of 21 computation- and storage-module. Each module consists of one intel Xeon E3-1220L V2 processor with two physical cores, 16GB DRAM module (Dell) and one SATA-SSD (Samsung). Each module has several PCI-express (PCIe) ports. Unlike SuperMikeII (traditional supercomputer) and SwatIII (regular datacenter), all the CeresII-module in a single chassis are connected to a common PCIe switch to communicate with each other. The high density of compute-modules per chassis in CeresII yields 42 physical cores connected through PCIe comparing to 16 physical cores per node as in SuperMikeII and SwatIII, thus resulting in better performance. Furthermore, the use of SSD reduce the io-wait and 8GB RAM per physical cores improves the access parallelism.

B. Hadoop configurations and optimizations

Since our goal is to evaluate the underlying hardwares and the balance among different hardware components (i.e. network, cpu-cores, storage and memory) 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 concurrent Yarn containers: We performed regorous testing on any of the clusters by launching different number of containers concurrently and reported the most optimized result.

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 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 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 each stage of PGA's 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. However, we did not find any visible performance difference (less than 2%) on any of the stages of our assembly pipeline due to change in the network. In order to investigate the reson in details, we measure the avaerage latency and the effective network-bandwith between two arbitrary compute nodes in SwatIII and SuperMikeII. Since SuperMikeII uses an Infiniband connection the average latency was found to be more than 10-times lower compare to SwatIII which uses ethernet. The avarage latency in SuperMikeII was found to be .014ms whereas in SwatIII, it is almost 0.2ms. However, due to the resource sharing among many users, the average effective-bandwith of SuperMikeII was found to be almost 10-times lower than that of SwatIII: 949Mbit/s in SuperMikeII, whereas 9.1Gbit/s in SwatIII. Furthermore, the java-based APIs of Hadoop or Giraph is not optimized to take the advantage of Infiniband. Hence, in any of the Hadoop or Giraph workloads in the moderate size bumble bee genome assembly pipeline we did not observe major performance difference using same number of nodes in SwatIII.

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 column of 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 50% improvement in this stage 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. As mentioned earlier, for any map-reduce job we always 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 during 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, the io-wait is reduced by using solid state drive (SSD) instead of HDD especially in case of shuffle-intensive Hadoop job.

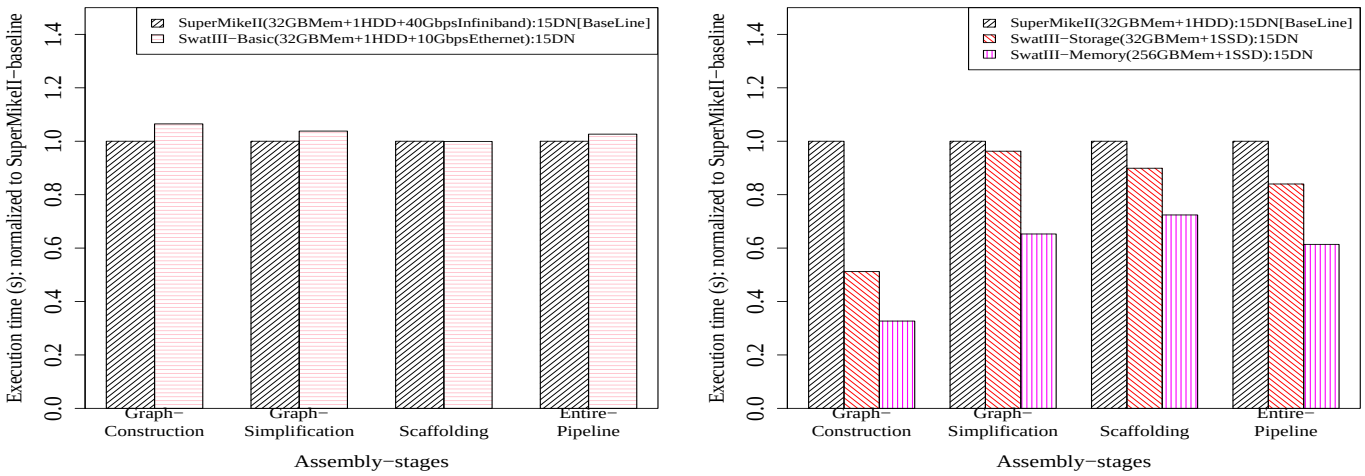
Figure-3 compares the read and write io operations per second to the local disk (of one datanode) for different stages of the same assembly using HDD and SSD. We observed almost 7 to 8 times improvement in the peak IOPS in case of SSD than HDD in the shuffle-intensive graph construction phase that writes 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



(a) Impact of network

(b) Impact of SSD and RAM

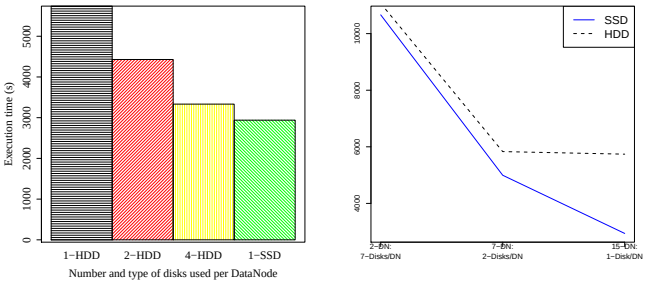
Fig. 1: Impact of each individual hardware component on different stages of the assembly pipeline (15 DataNodes used)

supersteps. Given enough memory, a huge amount of data is kept in cache and is fetched upon requirement during the next compute-superstep.

VIII. Performance Comparison between 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 merits of different cluster architecture in terms of raw execution time. Since most of the time the selection of the clusters are driven 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. the 15-DataNode 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) Because of the io-bound nature of the Hadoop job, SSD shows better scalability than HDD in the graph-construction stage with increase in number of nodes in the cluster, thus increasing the number of cores. There is almost no performance improvement in SwatIII-Medium-HDD (7-DN with 2-HDD each) and SwatIII-basic (15-DN with 1-HDD each). However, there is almost 50% improvement in the corresponding SSD case (i.e. SwatIII-medium-SSD and SwatIII-Storage). 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.

A. Scaledup cluster and SSD



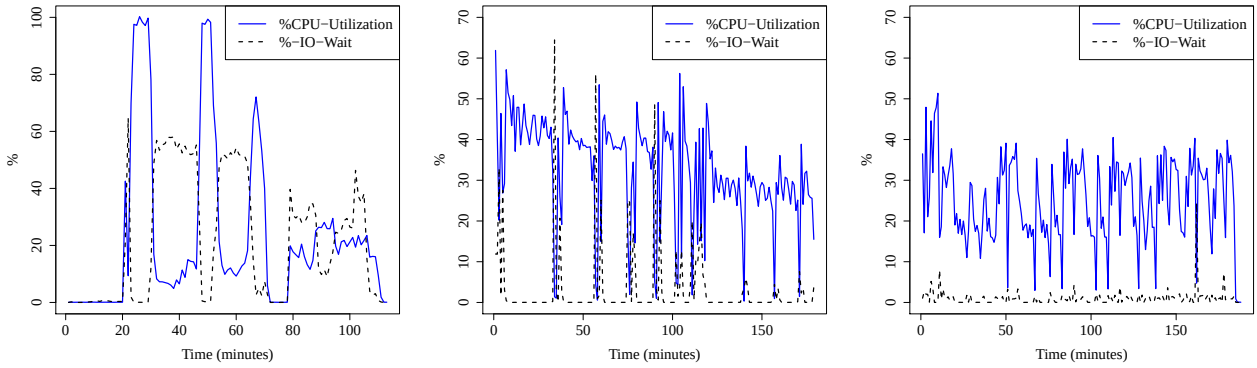
(a) Performance trend using 1, 2 and 4 HDD(s) and 1-SSD per node using a 15 datanodes

(b) Performance trend for SSD and HDD using 1, 2, 7 disks per node in 15, 7 and 2 datanodes

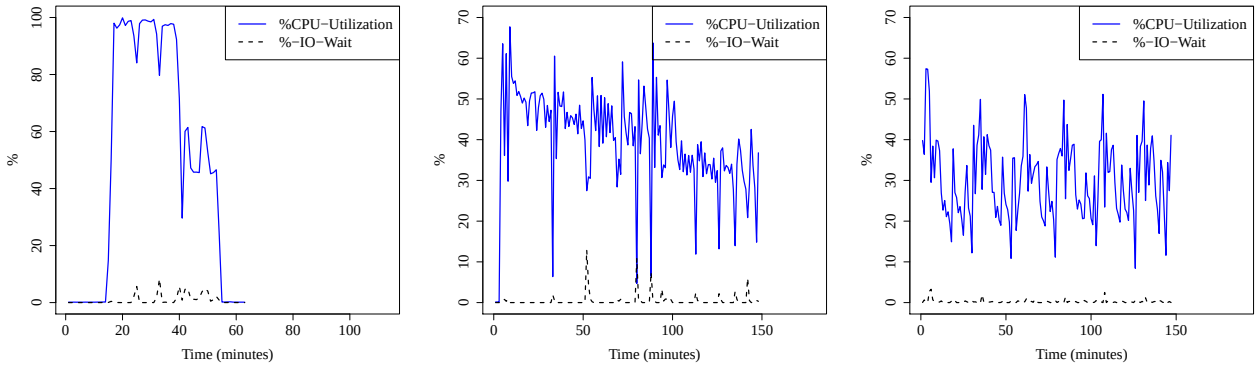
Fig. 6: Performance trend using HDD and SSD

Cloud service providers already started using SSDs as the elemental feature in their cloud infrastructure. Furthermore, many of the available cloud instances provide more SSDs per compute-node in order to improve the performance. Consequently, the setup-cost as well as the pricing of these scaledup instances increase. For example AWS i2.8xlarge storage-optimized instance offers 8SSDs per compute-node with 32 v-cores per node in a rate of $6.82 per hour which is one of the high-cost AWS-EC2-instances [3]. In this section, we analyze how to leverage SSDs in a cost-effective manner. In particular, we point out the scenario where HDDs and SSDs yield similar level of performance.

[3] http://aws.amazon.com/ec2/pricing/



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



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

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

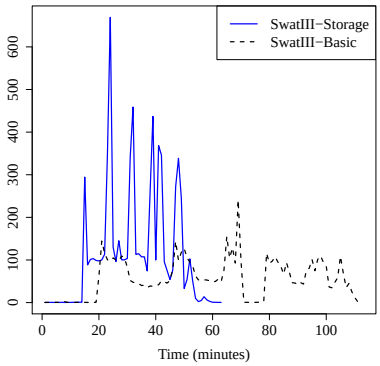
Figure-6a compares the performance of a single SSD and increasing number of HDDs per node for the Hadoop-based graph-construction stage of the bumble bee genome assembly pipeline. The performance improves almost linearly by increasing the number of HDDs per node in the cluster. On the other hand, 4-HDDs per node shows similar performance (only 5% variation) with a single-SSD per node. We did not observe any more significant improvement by providing more SSDs per node, concluding the disk-io rate reaches a saturation point. As a consequence, we did not find any significant performance-difference between SwatIII-FullScaleup-SSD and -HDD as shown in figure-6b while assembling the same bumble bee genome using 2-datanodes each with 7-disks. However, SSD shows significantly better performnce as well as scalability than HDD when we scale out by adding more compute nodes to the cluster (thus, increasing the total number of cores) and reducing the number of disks per node (thus, keeping the total storage space almost same in the cluster).

B. CeresII: Samsung-MicroBricks for shared nothing paradigm

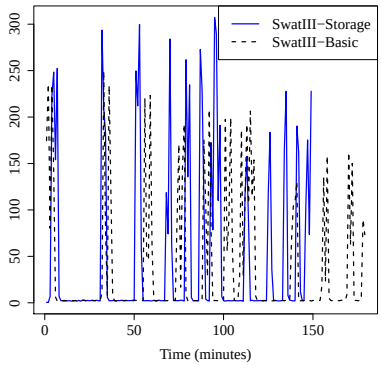
In this section, we evaluate a Samsung-MicroBricks based novel prototype-architecture called CeresII which is an improvement over CeresI [22]. As mentioned before, CeresII uses 2 physical cores, 1 SSD and 16GB memory per computation module. In order to assemble the 95GB bumblebee genome we use 32 such modules of this cluster as Hadoop-datanodes. The last columns of different stages of the assembly in Figure-7 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. Its performance is almost comparable to SwatIII-Medium-SSD which used 7-datanodes. However, the entire cluster (all the 32 modules) takes significantly less space than 8 nodes of SwatIII-Medium-SSD, thus yielding significantly better performance per rack-unit.

Component	Cost ($)
Intel SandyBridge Xeon 64bit Ep series (8-cores): used in SuperMikeII and Each variant of SwatIII	1520
Intel Xeon E3-1220L V2 (2-cores): used in CeresII	389
1-HDD (Western Digital, 500GB)	67
1-SSD (Samsung, 500GB)	177
Dell Poweredge 16GB meory module	139

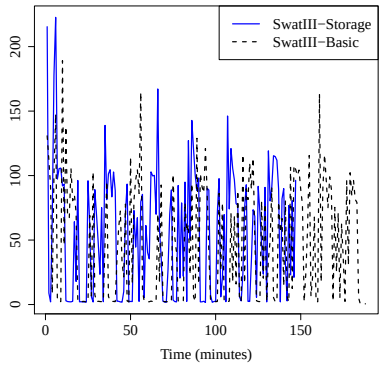
TABLE IV: Cost of each hardware component



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

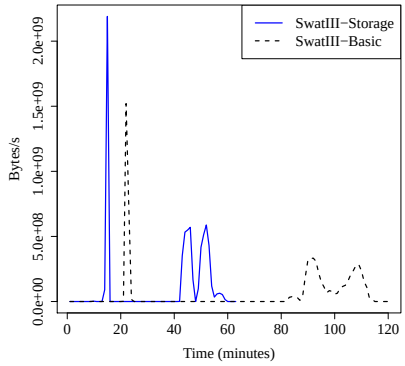


(b) Write-Iops in Graph Simplification in SwatIII-Basic and SwatIII-Storage

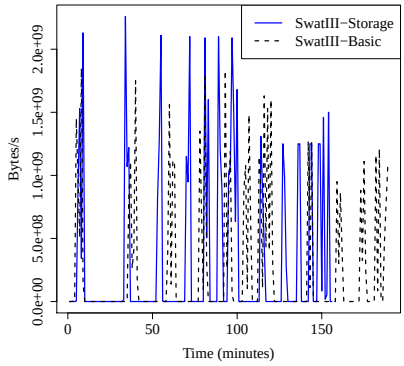


(c) Write-Iops in Scaffolding in SwatIII-Basic and SwatIII-Storage

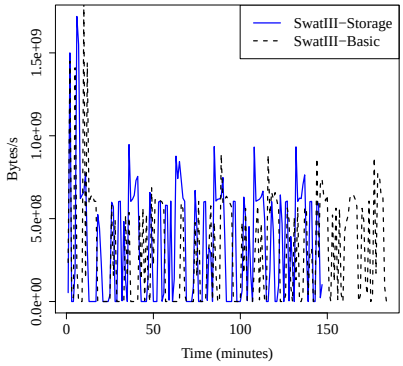
Fig. 3: Comparison of Total Write-IOPS in one host in SwatIII-Basic (1-HDD/node) and SwatIII-Storage (1-SSD/node)



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



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



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

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

IX. Price to Performance

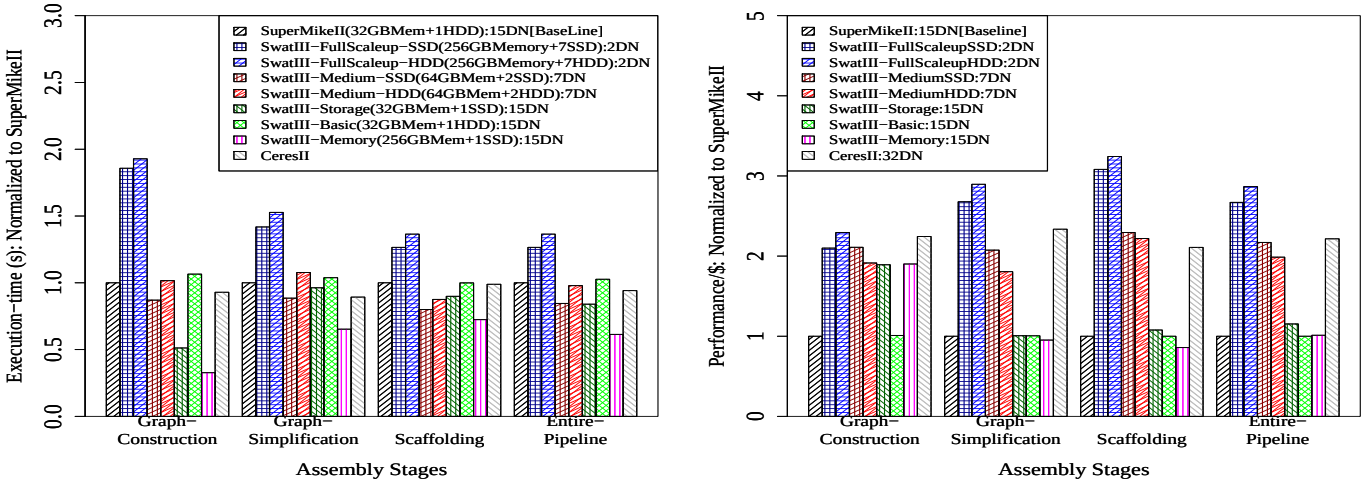
Table-IV show the cost of each hardware component used in different clusters. Price information is collected from [4] and [5] It is worthy to mention here, We do not assume that a single scaled up server with one disk can accomodate the entire data. The total amount of data should be held in its entirity 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.

[4]http://www.newegg.com/

[5]http://www.amazon.com/

X. Stress-testing of different clusters using human-genome

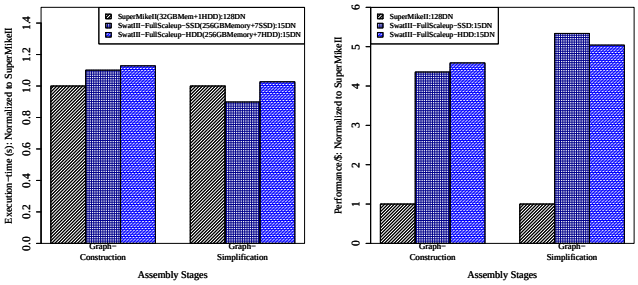
In order to evaluate the cumulative impact of all the hardware components (i.e. network, cpu-cores, storage and memory) on the entire cluster we summarize our study with a stress testing of SuperMikeII and SwatIII clusters using the large scale 452GB human genome that produces 3.2TB of graph(refer to table-II). In this part we utilize the maximum amount of resources that are available in any of the compute-clusters. We used 127-datanodes of SuperMikeII to accomodate the huge volume of data: either stored on disk (HDFS data or shuffled data) or the huge graph that is loaded in memory. On the other hand, we use 15-datanodes in the SwatIII-Full-Scaleup cluster (both HDD and SSD variant) which yields almost same amount of storage space as well as memory. As



(a) Execution-time (Lower is better)

(b) Performance/$ (Higher is better)

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



(a) Execution-time (Lower is better)

(b) Performance/$ (Higher is better)

Fig. 7: Compare different type of cluster architeture for human genome assembly pipeline

mentioned before, due to the low value of maximum allowed file descriptor in SuperMikeII ($ulimit - n$ is set to 1024) we could not run the scaffolding stage in this cluster. Hence, remove it from the comparison.

Figure-7a and 7b shows the execution time and the performance/$ respectively for the Hadoop-based graph-construction and Giraph-based graph-simplification stage of human genome assembly on three different cluster architectures. The observations are as follows: 1) The 128nodes of SuperMikeII (2032-cores) shows only 15-17% better performance than 15-datanodes of any variant of SwatIII-Full-Scaleup cluster (240 physical cores) while using almost 9-times more cores in the Hadoop-based graph-construction stage. The reason behind the suboptimal performance in superMikeII is two folded: first, the huge amount of io-wait resulted by only one HDD per node as discussed in section-VII-C. Second, the availability of less effective-network-bandwidth between compute-nodes because of resource sharing among many users as discussed in VII-B. However, Hadoop depends on network only once in the entire job: during data transfer tot he reducers 2) The Giraph-based graph-simplification stage performs 10% better in SwatIII-FullScaleup-SSD than SuperMikeII. Since Giraph is more network intensive, the higher effective-network-bandwidth between compute-nodes of SwatIII yields better performance although it has very less amount of cores than SuperMikeII. 3) Both HDD and SSD variants of SwatIII-FullScaleup shows almost similar performance for both Hadoop and Giraph (less than 5% variation). The effect is similar as discussed in section-VIII-A. 4) In terms of performance/$ any of the SwatIII-FullScaleup cluster is found to yield almost 4.5-times better result for the Hadoop stage and almost 5-times better result for the Giraph stage.

XI. Conclusion

Acknowledgment

The authors would like to thank...

References

[1] Z. Fadika, M. Govindaraju, R. Canon, and L. Ramakrishnan, "Evaluating hadoop for data-intensive scientific operations," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 67–74.

[2] S. Jha, J. Qiu, A. Luckow, P. Mantha, and G. C. Fox, "A tale of two data-intensive paradigms: Applications, abstractions, and architectures," in *Big Data (BigData Congress), 2014 IEEE International Congress on*. IEEE, 2014, pp. 645–652.

[3] S. Krishnan, M. Tatineni, and C. Baru, "myhadoop-hadoop-on-demand on traditional hpc resources," *San Diego Supercomputer Center Technical Report TR-2011-2, University of California, San Diego*, 2011.

[4] J. Vienne, J. Chen, M. Wasi-Ur-Rahman, N. S. Islam, H. Subramoni, and D. 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*. IEEE, 2012, pp. 48–55.

[5] J. Yu, G. Liu, W. Hu, W. Dong, and W. 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*. IEEE, 2013, pp. 708–713.

[6] Y. Kang, Y.-s. Kee, E. L. Miller, and C. Park, "Enabling cost-effective data processing with smart ssd," in *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*. IEEE, 2013, pp. 1–12.

[7] D. Wu, W. Luo, W. Xie, X. Ji, J. He, and D. Wu, "Understanding the impacts of solid-state storage on the hadoop performance," in *Advanced Cloud and Big Data (CBD), 2013 International Conference on*. IEEE, 2013, pp. 125–130.

[8] S. Moon, J. Lee, and Y. S. Kee, "Introducing ssds to the hadoop mapreduce framework," in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. IEEE, 2014, pp. 272–279.

[9] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Project Website*, vol. 11, no. 2007, p. 21, 2007.

[10] B. Li, E. Mazur, Y. Diao, A. McGregor, and P. Shenoy, "A platform for scalable one-pass analytics using mapreduce," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 2011, pp. 985–996.

[11] K. Krish, A. Khasymski, G. Wang, A. R. Butt, and G. Makkar, "On the use of shared storage in shared-nothing environments," in *Big Data, 2013 IEEE International Conference on*. IEEE, 2013, pp. 313–318.

[12] W. Tan, L. Fong, and Y. Liu, "Effectiveness assessment of solid-state drive used in big data services," in *Web Services (ICWS), 2014 IEEE International Conference on*. IEEE, 2014, pp. 393–400.

[13] S. Huang, J. Huang, Y. Liu, L. Yi, and J. Dai, "Hibench: A representative and comprehensive hadoop benchmark suite," in *Proc. ICDE Workshops*, 2010.

[14] M. Michael, J. E. Moreira, D. Shiloach, and R. W. Wisniewski, "Scale-up x scale-out: A case study using nutch/lucene," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. IEEE, 2007, pp. 1–8.

[15] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron, "Scale-up vs scale-out for hadoop: Time to rethink?" in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 20.

[16] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[17] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[18] T. Cheatham, A. Fahmy, D. Stefanescu, and L. Valiant, "Bulk synchronous parallel computinga paradigm for transportable software," in *Tools and Environments for Parallel and Distributed Systems*. Springer, 1996, pp. 61–76.

[19] A. Matsunaga, M. Tsugawa, and J. Fortes, "Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications," in *eScience, 2008. eScience'08. IEEE Fourth International Conference on*. IEEE, 2008, pp. 222–229.

[20] S. L. Salzberg, A. M. Phillippy, A. Zimin, D. Puiu, T. Magoc, S. Koren, T. J. Treangen, M. C. Schatz, A. L. Delcher, M. Roberts *et al.*, "Gage: A critical evaluation of genome assemblies and assembly algorithms," *Genome research*, vol. 22, no. 3, pp. 557–567, 2012.

[21] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[22] J. Min, H. Ryu, K. La, and J. Kim, "Abc: dynamic configuration management for microbrick-based cloud computing systems," in *Proceedings of the Posters & Demos Session*. ACM, 2014, pp. 25–26.