Journal of Computer and Communications, 2015, 3, 107-117
Published Online May 2015 in SciRes. http://www.scirp.org/journal/jcc
http://dx.doi.org/10.4236/jcc.2015.35014

Building a Productive Domain-Specific Cloud for Big Data Processing and Analytics Service

Yuzhong Yan[1], Mahsa Hanifi[1], Liqi Yi[2], Lei Huang[1]

[1]Department of Computer Science, Prairie View A&M University, Prairie View, TX, USA
[2]Intel Corporation, Hillsboro, OR, USA
Email: yyan@student.pvamu.edu, mhanifi@student.pvamu.edu, yiliqi@gmail.com, lhuang@pvamu.edu

Received March 2015

Abstract

Cloud Computing as a disruptive technology, provides a dynamic, elastic and promising computing climate to tackle the challenges of big data processing and analytics. Hadoop and MapReduce are the widely used open source frameworks in Cloud Computing for storing and processing big data in the scalable fashion. Spark is the latest parallel computing engine working together with Hadoop that exceeds MapReduce performance via its in-memory computing and high level programming features. In this paper, we present our design and implementation of a productive, domain-specific big data analytics cloud platform on top of Hadoop and Spark. To increase user's productivity, we created a variety of data processing templates to simplify the programming efforts. We have conducted experiments for its productivity and performance with a few basic but representative data processing algorithms in the petroleum industry. Geophysicists can use the platform to productively design and implement scalable seismic data processing algorithms without handling the details of data management and the complexity of parallelism. The Cloud platform generates a complete data processing application based on user's kernel program and simple configurations, allocates resources and executes it in parallel on top of Spark and Hadoop.

Keywords

Cloud Computing, Spark, Seismic Data Processing, Garbage Collection

1. Introduction

Cloud computing as a disruptive technology, provides a dynamic, elastic and easy-to-use computing climate to tackle the challenges of big data processing and analytics. Three different services cloud can provide in this regard, which are categorized as Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [1]. A cloud-based big data analytics platform is becoming important to support their daily work by delivering the required storage space, processing power, and intelligent analytics capacity in many industries, such as retails, energy, oil & gas, security/surveillance, image/video, social networks, financial/trading, and more. One challenge these industries are facing in common is the fast-growing data volume. The traditional HPC platform focusing on increasing FLOPS will need to be revisited to shift the emphasis to the data throughput and management instead.

In this paper, we studied the oil & gas industry requirements for the domain data processing and analytics, and then designed a domain-specific big data processing and analytics cloud for the industry. The oil & gas industry is a traditional domain that demands both high performance computing and big data storage to process large petroleum domain data, mostly seismic data. Seismic data analysis that processes and interprets multi-dimensional seismic volumes plays a key role in oil & gas exploration. The seismic data processing is both computation- and data-intensive, and is typically operated on top of traditional High Performance Computing (HPC) platforms. The size of seismic data, however, is increasing dramatically nowadays, which requires a new design for the data processing platform. Will the fast-growing seismic data benefit from the big data analytics platform and cloud computing techniques? How is the typical performance/scalability of using such a cloud? What are advantages/disadvantages comparable with traditional HPC programming?

The objective of the paper is to have a first attempt to explore and demonstrate the scalability and productivity of using the big data and cloud computing techniques for seismic data processing. In order to achieve the goal, a seismic analytics cloud (SAC) combining both big data platform and cloud computing is created to deliver a domain-specific Platform as a Service (PaaS) to support seismic data storage, processing, analytics and visualization. We have created a variety of seismic processing templates to simplify the programming efforts in implementing scalable seismic data processing algorithms by hiding the complexity of parallelism. The Cloud environment will generate a complete big data application on top of Spark based on user's kernel program and configurations, and deliver the required cloud resources to execute the application.

In following sections, we explain related works and background on different big data analytics platforms in Section 2. Section 3 follows by the design and implementation of the domain-specific cloud. We then present the performance details of a few case studies for seismic data processing in Section 4 and give performance analysis in Section 5. Finally conclusion and future works are discussed.

2. Related Work

We describe a few related works in this section, which are also our building foundations of the domain-specific cloud.

2.1. Apache Hadoop

Hadoop [2] with MapReduce [3] is the widely used open source framework in cloud computing for storing and processing large amount of data in the scalable fashion. There have been many studies [4]-[6] around performance of Hadoop on big data analysis. Hadoop with its ecosystem has been successfully deployed in many fields that require to process big data in batch processing. Hadoop File System (HDFS) supports distributed file system with fault tolerance feature, which provides a large, global-view, distributed file storage using loosely connected computing node disks together. MapReduce as the main parallel programming model provides a simple but typical parallel execution model that works well for applications with map-followed-by-reduce parallel execution pattern.

2.2. Apache Spark

Spark [7] [8] is the latest parallel computing engine working together with Hadoop that exceeds MapReduce performance via its in-memory computing and high level programming features. Spark is developed using Scala [9], which is a high-level programming language that supports both functional and object oriented programming. Comparable to DryadLINQ [10] Spark is equipped with an integrated environment for programming languages. Spark created a unique data structure called Resilient Distributed Datasets (RDDs) [11], which allows Spark application to keep data in memory, while MapReduce relies on HDFS to keep data consistent. RDD supports coarse grained transformation and logging them to provide fault tolerance. In time of losing a partition RDD can re-compute information using named logs to retrieve lost dataset [11]. Based on RDD, Spark supports more parallel execution operations than MapReduce. Defining RDDs via transformations and using them in various operations is the process of programming in Spark. Since transformations are lazy in Spark they won't compute till they are needed [12]. Moreover, Spark supports three high-level programming languages: Scala, Python and Java, while MapReduce only supports Java. Besides batch processing, Spark also supports streaming and interactive programming, which dramatically attracted the interests of many real-time and analytics applications developers. Spark community is very active in development, and Spark is quickly getting popular due to its unique features. The implementation and experiments of this paper are built on top of Hadoop and Spark environment.

3. Seismic Analytics Cloud Implementation

The goal of the seismic analytics cloud (we named it SAC) is to deliver a scalable and productive cloud Platform as a Service (PaaS) to seismic data analysis researchers and developers. SAC is designed to be able to store large amount of seismic data with major vendor's formats, as well as be able to process them in the scalable fashion to meet the performance requirements. Users should be able to work on their seismic processing algorithms using high-level programming models with very limited knowledge in parallelism and architecture.

3.1. The Architecture of Seismic Analytics Cloud

The design of SAC architecture is to emphasize twofold: one is to provide a high-level productive programming interface to simplify the programming efforts; the other is to execute user's applications with scalable performance. To achieve the first goal, we provide the web interface in which user could manage seismic datasets, programming within a variety of templates, generate complete source codes, compiling and then running the application and monitoring the job running status in SAC.

The interface allows users to write seismic data processing algorithms using our extracted common seismic computation templates, which lets users focus on their kernel algorithm implementation, and simplifies user's implementation in handling seismic data distribution and parallelism.

While the most popular-used programming models in seismic data processing include MATLAB, Python, C/C++, Fortran, Java and more, SAC supports Java, Python and Scala natively, so that users can write their own processing algorithms directly on our platform with these three languages; For legacy applications written in other languages, SAC uses the so-called PIPE mode to handle input and output data as standard-in and -out, which requires minor modifications of program source code on handling input and output.

SAC will generate complete Spark codes based on user's kernel codes and configurations, and then launch and monitor it on the SAC environment.

In order to support large amount data storage and scalable I/O performance, we chose Hadoop HDFS as the underlying file system, which provides fault tolerance with duplicated copies and good I/O throughput by supporting data locality information to applications. HDFS supplies out-of-the-box redundancy, failover capabilities, big data storage and portability. Since the size of seismic data is very large and keeps increasing constantly, HDFS provides a good solution for the data storage with fault tolerance property.

We use Spark as the parallel execution engine to start applications, since Spark works well on top of HDFS, Mesos [13] and YARN, and it provides a big data analytics computing framework with both in-memory and fault-tolerance support. Spark provides RDD as a distributed memory abstraction that lets programmers perform in-memory computations on large-scale cluster/cloud in a fault-tolerant manner. To get better performance, we need to put frequently used data into memory and processing data in memory, which is one key performance boost comparing with MapReduce. Some other useful packages and algorithms in data analytics, such as SQL, machine learning and graph processing, are also provided in Spark distribution version. We also integrated some common used libraries for image processing and signal processing, such as OpenCV, Breeze and FFTW etc., to provide a rich third party of libraries support to speed up the development process. **Figure 1** shows the overall software stack used in SAC.

Figure 2 presents the overall architecture of SAC. Based on the SAC web interface, Users are able to upload, view and manage their seismic data, which are stored on HDFS. They can then create their application projects by selecting a template from a list of predefined templates to start their own programming. After selected dataset and processing pattern, writing codes and compiling successfully, users can configure the running parameters and then submit jobs to SAC. Job status could be monitored while job is running and running results could be checked after job is finished. On the SAC backend, a big seismic data file will be split into multi-partitions and be constructed into RDD, which will be processed by working threads that apply user's algorithm in parallel. After all data are processed, the output data will be saved back to HDFS.

3.2. Input Data and Redirection

The SEG Y (also SEG-Y) file format [14] is one of several standards developed by the Society of Exploration

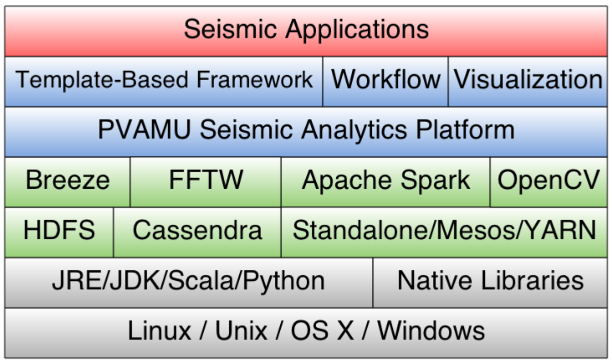


Figure 1. The software stack of seismic analytics cloud platform.

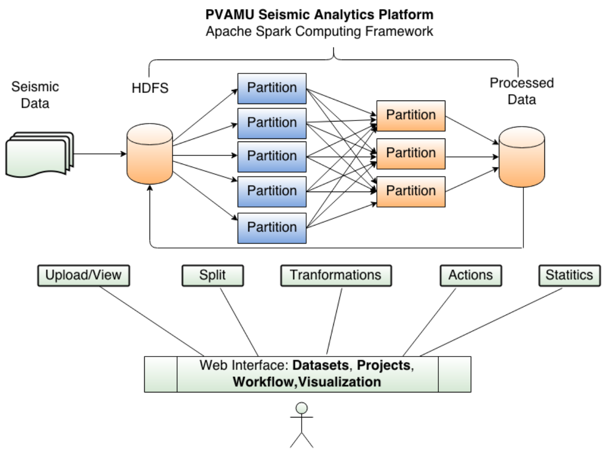

PVAMU Seismic Analytics Platform
Apache Spark Computing Framework
Seismic Data
HDFS
Partition
Processed Data
Upload/View
Split
Tranformations
Actions
Statitics
Web Interface: Datasets, Projects, Workflow,Visualization

Figure 2. The architecture of seismic analytics cloud platform.

Geophysicists (SEG) for storing geophysical data. This kind of big seismic data needs to be split into multiple small partitions to be processed in parallel. However, SEG Y data could not be split directly due to its irregularity, so we preprocess the SEG Y data format into a regular 3D volume data, and store the important header information into one xml file. Then the 3D volume data and xml will be feed into Spark applications. Spark uses InputFormat, which is the base class inherited from Hadoop to split such data and construct RDD. Each split will be mapped to one partition in RDD. The embedded InputFormat classes could not handle binary seismic data, so we implemented SeismicInputFormat in this project. Based on configuration defined by user while creating project, such as how many lines each split and number of overlap lines, SeismicInputFormat could spilt the 3D volume and feed partition to each mapper. The data of 3D volume is stored trace by trace in the Inline direction by default. For some algorithms that need to process data in cross-line or time-depth direction, we also provide interfaces to transform Inline format RDD into cross-line or time-depth direction. In this way, we could cache Inline format RDD in memory, thus all the transformations could be executed in memory with better performance.

3.3. Parallel Processing Templates for Seismic Data

Based on the general parallel execution patterns of seismic processing algorithms and applications, we predefined some templates to make this framework easy to program. Every template has explicit input type and output type. The typical templates are: Pixel pattern, which use sub-volume or one pixel as input and output one pixel; Line pattern, which treat one line as input and one line as output; SubVolume pattern, which feed user's application with sub-volume and get output from it in sub-volume format. A high level SeismicVolume class has been implemented in this project to provide user interface to access seismic volume. SeismicVolume class provides functions for constructing RDD based on processing templates user had selected, applying user's algorithms on RDD, and storing the final RDD on HDFS with format defined by user. To make it easy for programming, we provide some other functions to change the linear array into 2D matrix and 3D volume class; some functional programming interface such as iteration, map/flatMap, filter and zip could be used. We also integrated commonly used high-level algorithms, such as histogram, FFT, interpolating and filtering algorithms, so that user could put more attention on data analytics logic instead of details for each algorithm.

3.4. Code Generation

After user created project and completed their own kernel codes, one component named Code Generator (CG) in SAC will generate complete Spark codes for running on Spark platform. The CG will parse configuration of user's project and generate Spark application outlined codes, merge them with user's codes. User could also upload existing source codes or libraries, all of which will be integrated into current working project managed by Simple Build Tool (SBT). CG will also generate compiling and running scripts basing on user's runtime setting. All these scripts will be called by the web interface, on which some other information such as compiling and running status, location of output will be shown clearly.

3.5. Driver and Job Executor

In SAC, every user's project will be treated as one Spark application. CG will generate the main driver code for each project. Each application could be submitted to SAC for running after compiled successfully. At execution time, driver code will setup the Spark running time environment, call the SeismicVolume object to generate RDD and execute user's algorithms on top of RDD and then store the processed results on HDFS. It will clean up the running environment and release resources after finished. To make it support multiple users, Spark Jobserver [15] was introduced to this platform. Based on the priority of application and computation resources requirement of an application, an user could configure the running parameters: number of cores and memory size; and then submit his/her own job, monitoring job status and viewing the running results. Another big advantage of Spark Jobserver is supporting of NamedRDD that allows multiple applications share RDD but has only one copy cached in memory. For some complicate algorithms that need multiple steps or application running in workflow, NamedRDD is a good choice for boosting performance. After job is finished, the running results cloud be discarded or be saved to user's workspace basing on user's selection.

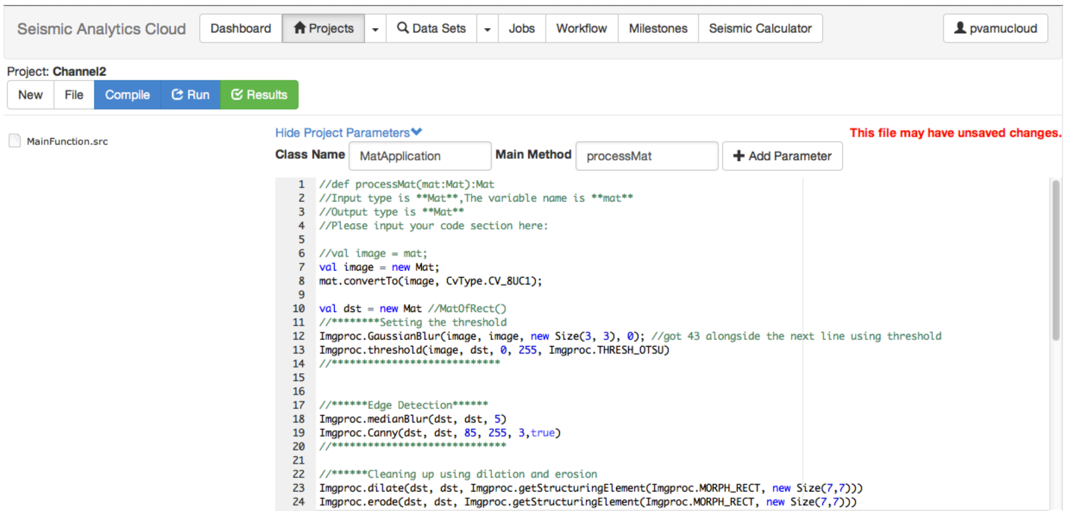
4. Experiment and Results

We have conducted numerous experiments on our 25 nodes of computer cluster located at Prairie View A&M University, in which one is master node and the other 24 are worker nodes. Each node of the cluster was configured with Intel Xeon E5-2640 Sandy Bridge CPU (2.5 GHz, 12 Cores), 64GB DDR3 memory. We have created a seismic data volume with 102GB, which is generated from the public Penobscot [16] seismic data from OpendTect [17] website with duplication and resampling. All of these experiments are performed with Spark 1.2.1 on Java 1.8.0 using different garbage collector setting [18] to be able to reduce garbage collection time as much as we can to improve the performance. Three test applications in seismic analysis are implemented and tested for the experiments: Seismic Calculator, Histogram, and Fast Fourier Transform (FFT). We have run these applications using different numbers of CPUs to show the scalability. We also changed the data split granularity to test performance impact: using 1 inline, 10 inlines, and 30 inlines per split.

All of these applications are tested in two ways: by running in Spark Shell using both cache option and un-cached one, and by submitting to Jobserver. We present the speedup by comparing with the corresponding sequential programs at the end. Spark performance web monitor UI, Spark Metrics and Nigel's performance Monitor (nmon) are used to observe detailed information about running times and performance of these tests. Nmon Analyzer [19] is used for following and observing cluster performance and finding the bottlenecks on the system. **Table 1** shows all results using various configurations in number of cores and splits.

4.1. SAC Web UI

Figure 3 shows the user interface of SAC. What user need for accessing seismic data hosted at cloud and verifying algorithm on it is only browser and an account. There are several tabs in SAC, such as Dashboard, Project,



```
//def processMat(mat:Mat):Mat
//Input type is **Mat**,The variable name is **mat**
//Output type is **Mat**
//Please input your code section here:

//val image = mat;
val image = new Mat;
mat.convertTo(image, CvType.CV_8UC1);

val dst = new Mat //MatOfRect()
//********Setting the threshold
Imgproc.GaussianBlur(image, image, new Size(3, 3), 0); //got 43 alongside the next line using threshold
Imgproc.threshold(image, dst, 0, 255, Imgproc.THRESH_OTSU)
//****************************


//******Edge Detection******
Imgproc.medianBlur(dst, dst, 5)
Imgproc.Canny(dst, dst, 85, 255, 3,true)
//*****************************

//******Cleaning up using dilation and erosion
Imgproc.dilate(dst, dst, Imgproc.getStructuringElement(Imgproc.MORPH_RECT, new Size(7,7)))
Imgproc.erode(dst, dst, Imgproc.getStructuringElement(Imgproc.MORPH_RECT, new Size(7,7)))
```

Figure 3. The SAC user interface.

Table 1. Running time for applications with various configurations (in seconds).

Application	Best Speedup	Split	No. of Cores		
			64	144	288
Calculator	**120**	1	36	19	15
		10	29	19	17
		30	56	44	46
FFT	**116**	1	132	66	58
		10	90	54	51
		30	108	66	58
Histogram	**115**	1	108	84	72
		10	228	240	270
		30	840	840	720

Datasets, Jobs, Workflow and some other useful tools. Dashboard will give user a brief view about how many projects he/she had created and usage statistics of cluster. In Project tab, user could create new project, edit existing project, compile and run project. Jobs tab will show status of all running and finish jobs. User could view data sets and select on them to analyze in Datasets tab. Workflow is designed for complicate algorithms or batch jobs but still provide flexibility and usability to user for configuration.

4.2. Seismic Calculator

Seismic calculation is a simple, useful but time consuming process when seismic data is big. In addition to the operations between two volumes, various types of arithmetic operations can be performed on a single seismic volume. These operations include arithmetic and logic ones that apply to every single sample in the volume.

4.3. Fast Fourier Transform (FFT)

FFT is the most popular algorithm for computing discrete Fourier transform (DFT), which is widely used in science and engineering. In seismic velocity model and image analysis, FFT is almost first and fundamental step. There are different implementations of FFT, such as FFTW, OpenCV, Kiss FFT, Breeze etc. Breeze is one of libraries in ScalaNLP, which includes a set of libraries for machine learning and numerical computing. Spark itself already includes Breeze in its release, so we choose FFT algorithm in Breeze for experiment.

4.4. Histogram

This is the third application used for performance analysis. Histogram is to compute the data range distribution, which is used for estimation of the probability distribution of continuous quantitative variable. It is also a basic method for seismic data analytics. Spark already provides function to get histogram information from RDD directly. The bin size we choose for experiment is 10.

5. Performance Analysis

From the experiment results shown in **Table 1**, the speedup of parallel codes is apparent. In this section, we will discuss the usability of SAC, and make deep performance analysis to find the bottleneck, which will also conduct performance tuning in the future.

5.1. Usability Analysis

In the traditional seismic data processing methods using HPC, the product development flow requires a lot of geophysicists and IT developers involved: verifying algorithm with small sample data at first, then transferring into MPI codes with parallel optimization to handle actual big data. The whole process is time consuming and low efficient, and sometime even lead in consistent results between experiment data and actual data. On SAC, geophysicists and data scientists could verify their algorithms and directly experiment them with actual data. SAC could handle data distribution, code generation and execute the application in parallel automatically, but could provide fault tolerance natively and scalability. Take the 2D FFT case as example, user only needs to select template, write FFT algorithm or call other existing APIs, and type this piece of codes in SAC, in such function the input plane and output plane are already defined by SAC. The only things left are selecting data sets, compiling and running application, then viewing the results. In short, user only needs to take care about algorithm, and SAC will handle most of others, thus improve productivity apparently.

5.2. Performance Analysis of Seismic Calculator

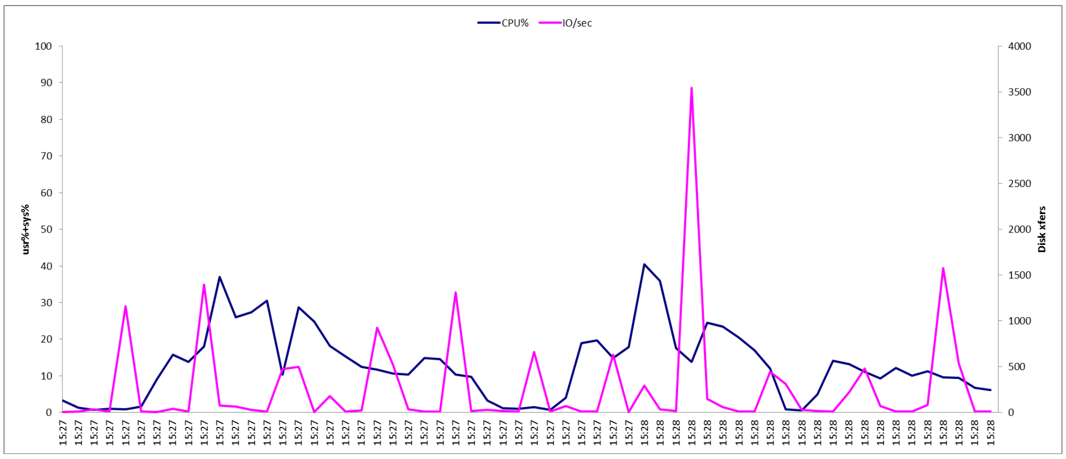
Among all three different number of split sizes, the best results for calculator is achieved with 288 cores in first two, which indicates that more computing resource could get better performance.

Closer look at the system with nmon-analyzer during run-time gives an interesting chart in network situation, CPU usage and the I/O of the system. **Figure 4** shows these data versus each other. **Figure 4(a)** shows CPU performance while on the other hand **Figure 4(b)** shows the network packets sending and receiving. It is obvious in the diagram that at the peak time for network CPU is not busy and at some points it became idle because of waiting for data. Increasing in network speed to have a better response for I/O request seems to be a key point in boosting the performance.

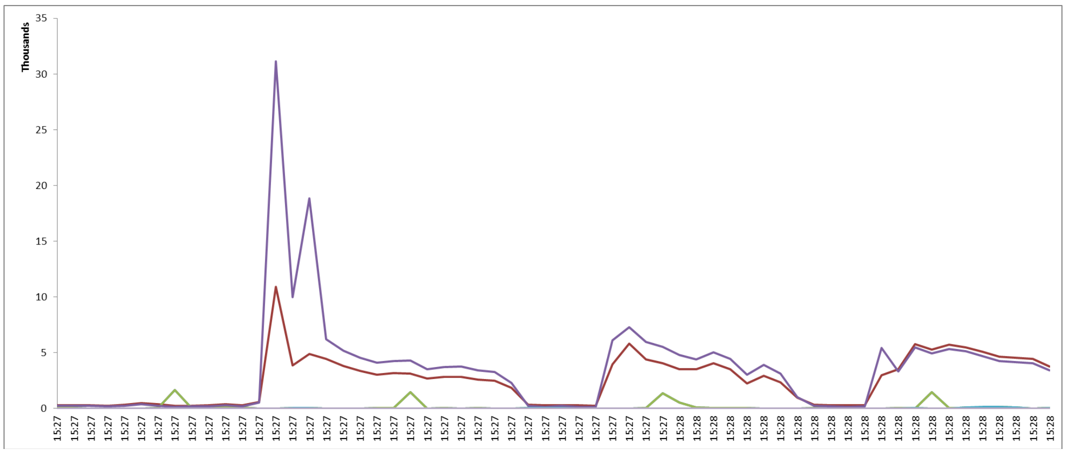
5.3. Performance Analysis of FFT

For FFT, it is a computing intensive workload hunger for CPU cycles instead of IO bandwidth. One system form the cluster was picked to show the performance characteristics in the run time. In **Figure 5(a)**, CPU utilization quickly ramps up to 95% user time and mostly stays at the same level with several dips till the end of execution. There was not much time spend in kernel mode or waiting for disk/network IO. There could be a little space for performance tuning to shorten the ramping up time in the start stage and remove the dips during the run. **Figure 5(b)** shows the disk read and write during the lifetime of the job. The maximum write is about 70 MB/s and the peak read is 50 MB/s. Both the read and write have not reached the bandwidth ceiling of the system. Same as the disk utilization, the network bandwidth was under 10 MB/s, which indicates underutilized network. The memory utilization in **Figure 5(d)**, shows that memory was 60% occupied by FFT.

The best results for this application are gained by using number of split size with 10 and number of cores 288. From the performance characteristics described earlier, FFT being a computing hunger workload, adding more computing power always will be beneficial, till other resources got over subscribed.



(a)



(b)

Figure 4. (a) CPU and I/O and (b) Network packets for calculator.

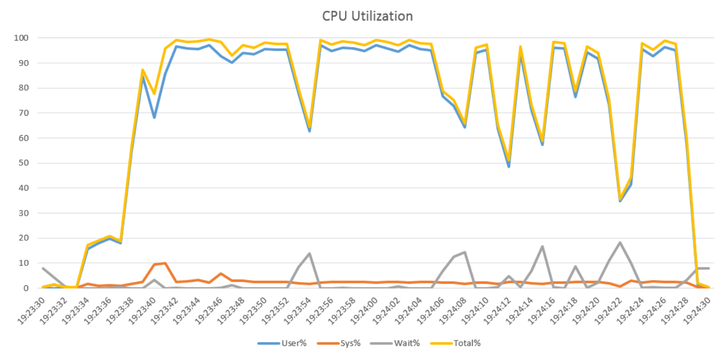
5.4. Performance Analysis of Histogram

Similar with FFT, Histogram could also be qualified as computing intensive workload, where computing power is the primary factor influencing performance. There are, however, two differences from FFT. First, Histogram has two distinct stages in the run time. The first stage last from the start of the run till the middle. In this stage, CPU utilization is considerably high with peak user time close to 90%. And the second stage is from the middle to the end. Where user time is below 30% most of the time with maximum a little above 50%. Second, in the more CPU cycle hunger stage, the CPU utilization is not as high as that in FFT, which was above 95% most of the time. System and IO wait is not high, indicating no bottle neck comes from IO or system activities. **Figure 6(b)** shows the disk read and write behavior. The disk was under utilized both for the read and write. And from **Figure 6(c)**, similar behavior with FFT case, traffic peaks below 10 MB/s. Memory got utilized more comparing to FFT, here the unused memory is 27%, as shown in **Figure 6(d)**.

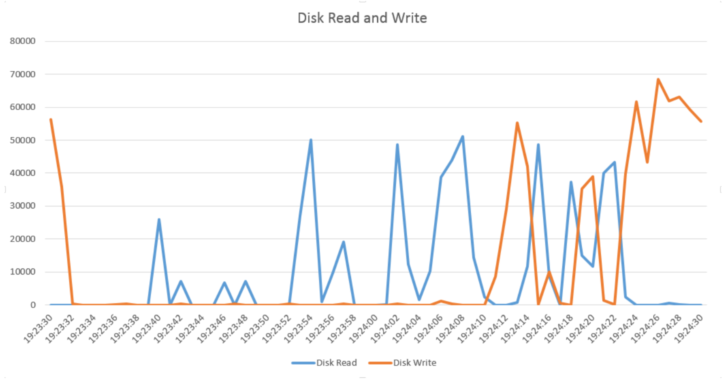
For Histogram, the best performance is also using small split. However, there is one case where 28MB data split is used and 144 core case out performs 288 core case. One possible reason could be explained by the larger memory foot print of this workload and the GC activities of Java virtual machine. As we all know, when processing larger data size with the same heap size, JVM has a tendency to get involved in longer GC pauses. And all the GC pauses will add up to the final run time, making the entire run slower.

6. Future Work and Conclusion

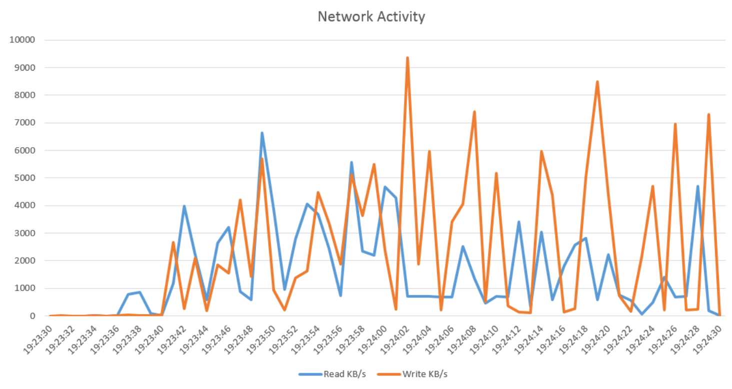
We continue to work on SAC to make it more productive in development and scalable in performance. One



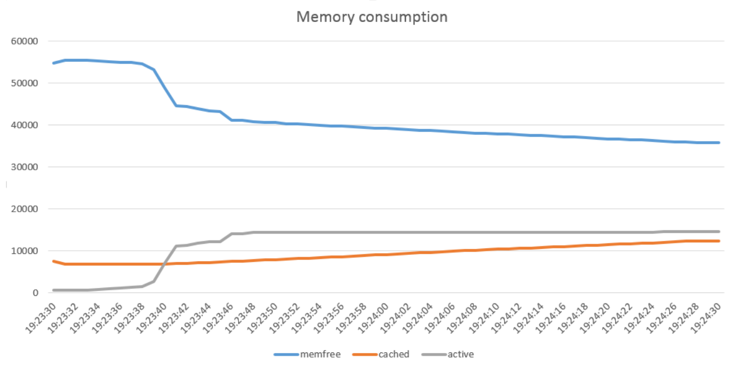
(a)



(b)

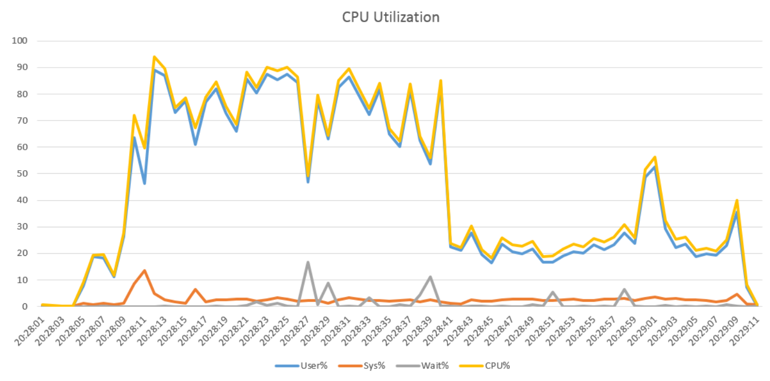


(c)

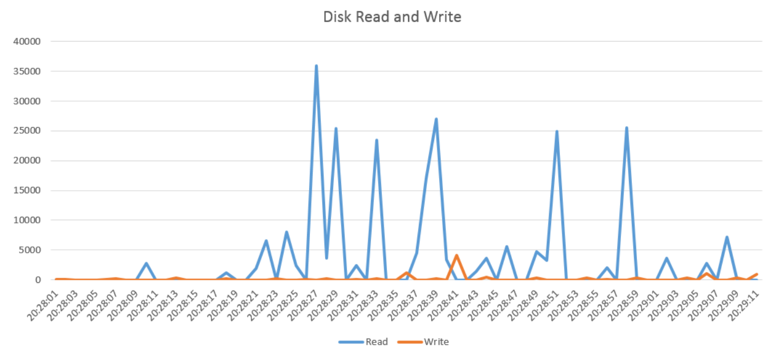


(d)

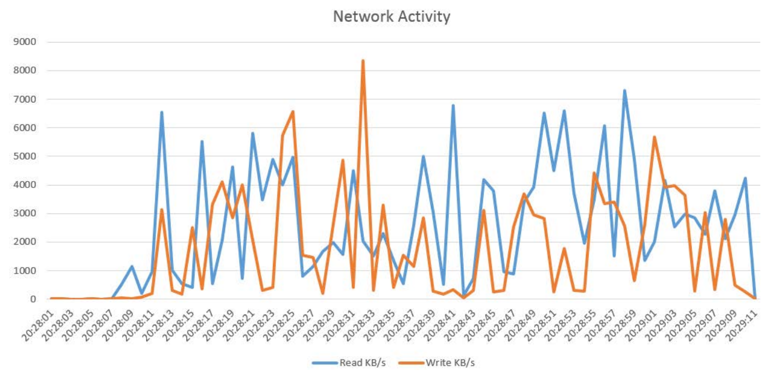
Figure 5. (a) CPU; (b) Disk I/O; (c) Network activity; (d) Memory utilization for FFT.



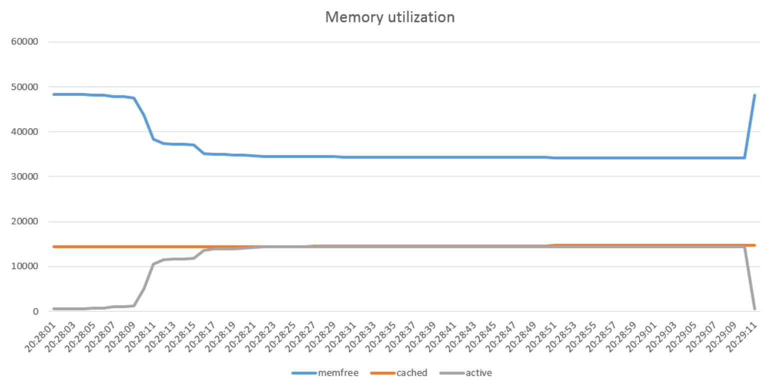
(a)



(b)



(c)



(d)

Figure 6. (a) CPU; (b) Disk I/O; (c) Network activity; (d) Memory utilization for histogram.

main focus we are working on is to provide a workflow framework to allow users to drag-and-drop to create a complete workflow by combining a sequence of programs. SAC will be able to connect them and launch in a single Spark context to reuse and keep data in memory as much as possible. We will also enhance the overall performance and visualization capability. We plan to open SAC to industry to collect more feedback for further improvement.

Acknowledgements

This project is supported in part by National Science Foundation CRI program award \#1205699.

Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Agrawal, D., Das, S. and El Abbadi, A. (2011) Big Data and Cloud Computing: Current State and Future Opportunities. *Proceedings of the* 14th *International Conference on Extending Database Technology*, *ACM*, 2011, 530-533. http://dx.doi.org/10.1145/1951365.1951432

[2] Hadoop Introduction (2014). http://hadoop.apache.org/

[3] Ghemawat, J.D.S. (2008) MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, **51**, 107-113. http://dx.doi.org/10.1145/1327452.1327492

[4] Islam, N.S., Rahman, M., Jose, J., Rajachandrasekar, R., Wang, H., Subramoni, H., Murthy, C. and Panda, D.K. (2012) High Performance RDMA-Based Design of HDFS over InfiniBand. *Proceedings of the International Conference on High Performance Computing*, *Networking*, *Storage and Analysis*, 35. http://dx.doi.org/10.1109/SC.2012.65

[5] Kim, K., Jeon, K., Han, H., Kim, S.-G., Jung, H. and Yeom, H.Y. (2008) Mrbench: A Benchmark for Mapreduce Framework. 14*th IEEE International Conference on Parallel and Distributed Systems*, 2008, 11-18. http://dx.doi.org/10.1109/ICPADS.2008.70

[6] Lu, X., Wang, B., Zha, L. and Xu, Z. (2011) Can MPI Benefit Hadoop and MapReduce Applications? 2011 40*th International Conference on Parallel Processing Workshops* (*ICPPW*), 2011, 371-379.

[7] Spark Lightning-Fast Cluster Computing (2014). http://spark.incubator.apache.org/

[8] Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S. and Stoica, I. (2010) Spark: Cluster Computing with Working Sets. *Proceedings of the* 2nd *USENIX Conference on Hot Topics in Cloud Computing*, Berkeley, 2010, 10. http://dl.acm.org/citation.cfm?id=1863103.1863113

[9] Odersky, M., Spoon, L. and Venners, B. (2008) Programming in Scala. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.202.9255n&rep=rep1n&type=pdf

[10] Yu, Y., Isard, M., Fetterly, D., Budiu, M., Erlingsson, U., Gunda, P.K. and Currey, J. (2008) Dryadlinq: A System for General-Purpose Distributed Data Parallel Computing Using a High-Level Language. *OSDI*, **8**, 1-4.

[11] Mosharaf Chowdhury, M.Z. and Das, T. (2012) Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. *NSDI*'12 *Proceedings of the* 9*th USENIX Conference on Networked Systems Design and Implementation*, San Jose, USENIX Association Berkeley, April 2012.

[12] Su, X., Swart, G., Goetz, B., Oliver, B. and Sandoz, P. (2014) Changing Engines in Midstream: A Java Stream Computational Model for Big Data Processing. *Proceedings of the VLDB Endowment*, **7**.

[13] Mesos: A Distributed Systems Kernel (2014). http://mesos.apache.org

[14] S. T. S. Committee (2002) SEG Y rev 1 Data Exchange Format.

[15] Spark Jobserver: REST Job Server for Spark (2014). https://github.com/ooyala/spark-jobserver

[16] Penobscot 3D-Survey (2015). https://opendtect.org/osr/pmwiki.php/Main/PENOBSCOT3DSABLEISLAND

[17] Free Open Source Seismic Interpretation Platform (2015). http://opendtect.org/

[18] Part #1-Tuning Java Garbage Collection for HBase (2015). https://software.intel.com/en-us/blogs/2014/06/18/part-1-tuning-java-garbage-collection-for-hbase

[19] nmon_analyser (2015). https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Power%20Systems/page/nmon_analyser