International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 6, Issue 5, September- October 2017 ISSN 2278-6856

A SURVEY ON A NOVEL SPARK ON HADOOP YARN FRAMEWORK BASED IN-MEMORY PARALLEL PROCESSING FOR EFFECTIVE PERFORMANCE

V.Sreedevi[1], J.Swami Naik[2]

[1]PG Student, Computer Science and Engineering Dept, GPREC, Kurnool (District), Andhra Pradesh-518004, INDIA.

[2]Associate Professor, Computer Science and Engineering Dept, GPREC, Kurnool (District), Andhra Pradesh-518004, INDIA.

Abstract
A Novel spark is extends the popular MapReduce model to efficiently support more types of computations, including interactive queries and streaming. Nowadays speed is important in processing huge datasets, as it means the difference between exploring data interactively and waiting minutes or hours. One of the main features Spark offers for speed is the ability to run computations in-memory, but the system is also more efficient than MapReduce for complex applications running on disk. In this paper we are facilitate implementation and assure high performance of spark based algorithms in a complex cloud computing environment, for a parallel programming model is used. By incorporating RS data with Resilient Distributed Datasets (RDDs) of spark, all level parallel RS algorithms can be easily expressed with transformations and actions. And also to improve the performance Data-intensive multitasking algorithms and iteration-intensive algorithms were evaluated on Hadoop YARN framework. By supporting these workloads in the same engine, Spark makes it easy and inexpensive to combine different processing types, which is often necessary in production data analysis pipelines. In addition it reduces the management burden of maintaining separate tools.

Keywords*:* Apache Spark, big data, Hadoop yet anotherresource negotiator (YARN), parallel processing, remote sensing (RS).

1. INTRODUCTION
Remote-sensing (RS) knowledge shows vital potential for users to explore heap of changes in patterns of activities in the current fast phased world. However, the large volume of high resolution RS knowledge can not be handled by one computing node. the speed of knowledge production has full current processor capacities. the range of RS knowledge means image process algorithms can not be created generic for additional analysis. two-faced with such RS "big data" challenges, use of a parallel programming model like MapReduce is required for international analysis. Current work concentrates on special algorithms, and models demonstrate restricted price in fast the knowledge-discovering method. Only a few operational comes involve a way to parallel method large RS knowledge with generic programming models. Sensible cases that reveal performance gains on corresponding platforms are rather scarce to the present finish, we tend to here answer the essential question of how to design generic parallel algorithms which will be expeditiously executed on candidate platforms for this work.

In nowadays, the development of sensor technology leads to higher resolution and increasing quantity of the data of images. These changes mentioned above make traditional methods of image change detection more and more inefficient and powerless for larger scale images. In order to meet the challenge, high performance computing systems have been attracting more and more interests of researchers. High Performance Computing (HPC) systems include cluster and specialized hardware (GPUs and FPGAs). Several efforts have been made to take advantage of the HPC systems in image processing and other applications with large scale data. [1-2] make use of the extensive computational power of the GPU (Graphics Processing Unit) to speed up their algorithms. But the capability of memory in GPU is not large enough for addressing large scale images, and the programming model for GPU is complicated usually. [3] uses MPI (Message Passing Interface) to transmit message between cluster nodes to run their algorithm distributed. However, MPI has a low fault-tolerant performance, and its programming is also complicated. [4] explores rapid processing methods and strategies for RS (remote sensing) images based on Hadoop, an open resource cloud computing framework, which is deployed on a cluster and automatically schedules the nodes of the cluster working collaboratively. Hadoop implements MapReduce, a programming model proposed by Google for processing and generating large datasets with a parallel, distributed algorithm on a cluster. Nevertheless, MapReduce is designed for acyclic data flow model. Most algorithms for image processing are iterative models and reuse a working set of data across each iteration. These two mismatches lead to low performance when migrate such image processing algorithms to MapReduce model compared with processing an acyclic data flow. Spark is designed for the focus on applications, that reuse a working set of data across multiple parallel operations, which include two use cases that are inefficient on MapReduce: iterative jobs and interactive analytics.

The rest of this paper is organized as follows. The Hadoop YARN platform and Apache Spark are briefly introduced in Section II, after which implementation of a generic parallel programming model and universal logical image partition method are described in Section III. In Section IV, the experimental environment and results of the case and algorithms use are introduced, followed with a discussion in Section V. Finally, conclusion and future work are introduced in Section VI.

2. USING SPARK ON HADOOP YARN

Spark need the resources to do the computation and Yarn manage all the resource (CPU cores and memory) on the cluster. Understand how Spark communicate with Yarn to finish all the tasks could help us better analyse the different Spark problem. Hadoop Yarn has 3 components: Resource Manager (RM), Application Master (AM) and Node Manager (NM). Spark on Yarn support 2 deploy mode, “cluster” and “client”, the only different is your SparkDriver running inside or outside of a Yarn Container, cluster mode, which is normally used on the production system. The code source is base on Spark 2.0. When start a Spark application by spark-submit command, in SparkSubmit (1k line size static class), on cluster mode, the class org.apache.spark.deploy.yarn.Client will be executed. The first thing is to submit a Yarn application to Yarn Resource Manager (RM). On more detail, this include creating, starting a YarnClient object from Apache Yarn library, and calling its createApplication() method. With first received GetNewApplicationResponse instance from Yarn, we check whether the max memory resource capability is enough for launch the Spark Driver and Executors. Then it create the ContainerLaunchContext (all the JVM options are here, also specific the AM executable class name org. apache. spark. deploy. yarn. Application Master), and the Application Submission Context (app’s tags, max attempts, capability required), call submit Application() of Yarn Client to finally submit the new application. In the end, waiting for RM to launch the AM on the first allocated container. In the run() method of AM, it will start a single Thread (called DriverThread) to execute the user class specific in—class JVM parameter (this is the Main class of your Spark application). Spark application Driver will be initialised in this Thread. A lot of checks and initialisations will happen when the Spark Context has be created. The main different with Yarn is in create Task Scheduler() of Spar Context class, it will return one instances of Yarn Scheduler and one Yarn Scheduler Backend. Here the YarnScheduler is very similar to the Task Scheduler Impl, but the Yarn Scheduler Backend is the bridge between the Yarn containers and Spark tasks.
Continue inside of AM, AM register itself to Yarn RM Client, and get back a instance of YarnAllocator. Spark Yarn RM Client is just a wrapper for the the Yarn’s AMRM Client to handle the registering and unregistering the AM with Yarn RM. Once AM got this Yarn Allocator, first thing is call its allocateResources() method, this will use Yarn’s AMRM Client to allocate all the containers (executors) for the first time (all the target Num Executors are come from the SparkConf). Next major thing is to launch a ReporterThread which keeping call allocateResources() and do some sleep. For each container allocated successfully, in the runAllocatedContainers() method of Allocator, it will launch one instance of ExecutorRunnable to start the Executor in this container. ExecutorRunnable prepare the container’s command(run java command for CoarseGrainedExecutorBackend class), set up the container’s environment, and call startContainer() of Yarn NMClient to finally start an Executor in the container. Spark use RM, AM and NM to start the Driver and Executors in Yarn containers. The main containers’s handling work is done by YarnAllocator. Allocator has the status of all the running and failed Executors, its allocateResources() method will be called by AM periodically. YarnAllocator also deal with this PreferredLocalities, handle the completed containers, keeping update the resource request for RM Manage all the running containers for a Spark application is a busy job

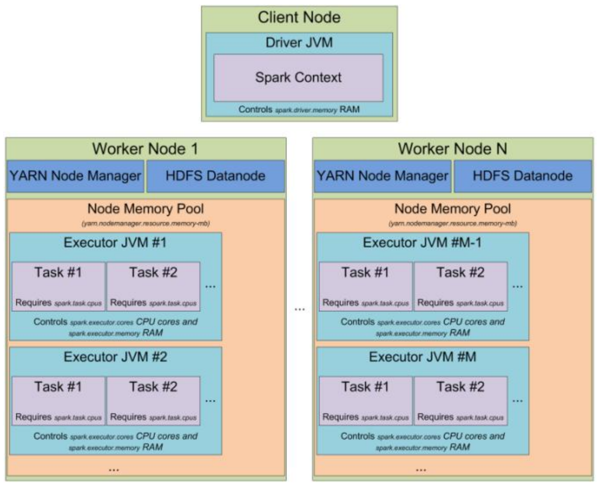


Fig. 1. Architecture of Spark YARN model.

3. RELATED WORK

A. NaiveBayesModel

Naive Bayes is a simple multiclass classification algorithm with the assumption of independence between every pair of features. Naive Bayes can be trained very efficiently. Within a single pass to the training data, it computes the conditional probability distribution of each feature given label, and then it applies Bayes’ theorem to compute the conditional probability distribution of label given an observation and use it for prediction.

MLlib supports multinomial naive Bayes and Bernoulli naive Bayes. These models are typically used for document classification. Within that context, each observation is a document and each feature represents a term whose value is the frequency of the term (in multinomial naive Bayes) or a zero or one indicating whether the term was found in the document (in Bernoulli naive Bayes). Feature values must be nonnegative. The model type is selected with an optional parameter “multinomial” or “bernoulli” with “multinomial” as the default. Additive smoothing can be used by setting the parameter λλ (default to 1.01.0). For image classification, the input feature vectors are usually sparse, and sparse vectors should be supplied as input to take advantage of sparsity. Since the training data is only used once, it is not necessary to cache it.

B. SVMModel

The linear SVM is a standard method for large-scale classification tasks. It is a linear method as described below in equation, with the loss function in the formulation given by the hinge loss:

$$L(w;x,y):=\max\{0,1-yw^Tx\}$$

By default, linear SVMs are trained with an L2 regularization. We also support alternative L1 regularization. In this case, the problem becomes a linear program.

The linear SVMs algorithm outputs an SVM model. Given a new data point, denoted by xx, the model makes predictions based on the value of wTxwTx. By the default, if wTx≥0wTx≥0 then the outcome is positive, and negative otherwise.

C. RS Datasets

For this experiment, we have taken continuous remote sensor images from various test drive cars as described below.
Dataset -1Name: Traffic Remote Sensor Images
Size: 130GB
Number of images: 30,249
Image Resolution: 1280 x 960
Dataset -2 Name: Test Drive Remote Sensor Images
No. of Sensors: 6
Full dataset Size: 233GB
Number of images: 7200 per each sensor
Total No. of images: 7200x6=43,200
Image Resolution: 1280 x 960

1277381619Image000012.jpg
1277381619Image000013.jpg
1277381619Image000014.jpg
1277381619Image000015.jpg

Fig 2: Sample data for testdata-1

D. Implementation Model

Since RDD can only be created from stable storage such as HDFS or other RDDs, Spark-based algorithms that process RS data stored in HDFS suffer the impacts of image partition scale. Scale is not only related to efficiency in generating key/value records, but also the scale is highly correlated with algorithms for implementation and fault-tolerant cost recovery. Al the RS images are processed using spark with RDD. Suppose that the total N megabytes of RS data are stored in HDFS and the default HDFS block size is M megabytes, and that the HDFS replication factor is denoted as factor. Let disk I/O rate be V MB/s, the pixel numberof a RS image be proportional to the size of the RS image, and each pixel be I bytes. Suppose that there are H hosts and each has C MB/s processing speed. Assuming scheduling costs are negligible, the total processing time T for the simplest operations such as inversion of the pixel digital number can be formulated as following:

$$T=\frac{\frac{N}{M}\times\left(\frac{M}{V}+\frac{2^{20}\times M}{IC}\right)}{factor\times HC}$$

where the numerator represents the time required to read all pixels from HDFS, and the denominator represents the sum of processing capacities of the underlying hardware. The two components in the bracket of the numerator represent time cost to read data, and time cost for key/value records gener-ation, respectively. As shown in the above formula, the time cost of key/value records generation is unrestricted.

4. RESULTS AND ANALYSIS

A. Experiment Environment

The experiment environment consisted of three nodes. We used 1 Dell PowerEdge inspiron server, with 500 GB disk. We employed an Oracle Virtual Box 5.0.0 to create three virtual nodes. To monitor cluster performance, a Ganglia monitor (core 3.6.0) is used [48]. The first three nodes in Table I adapt as OS Ubuntu 14.04.2 LTS, while the others adapt Ubuntu 12.04.5 LTS. The Hadoop version used is hadoop-2.4.0. The Spark version is Spark 1.4.0. The Scala interpreter version is Scala 2.10.4. The JDK version is JDK 1.7.0_45. The cluster is heterogeneous with different hardware (servers, workstations, and PCs), different networks (two subnetworks), and different operating systems (Windows and Linux). Computing nodes having better performance are configured with less disk space, while nodes with fewer CPUs are configured with more storage disks. To optimize the execution of Spark applications on a Hadoop YARN model, parameters were tuned according to Cloudera's Guide [49]. Specifically, we preserved the necessary memory space for OS and its daemon services using "yarn. nodemanager. resource. memory-mb" to limit the amount of physical memory, in MB, that can be allocated for containers. Also configured parameters are "yarn.scheduler.minimum-allocation-mb", yarn.app.mapreduce.am.resource.mb" which control the smallest amount of physical memory, in MB, that can be requested for a container and the physical memory requirement, in MB, that can be allocated for AMs, respectively. YARN container were set with at least 2 GB physical memory. AMs were configured with at least 4 GB memory by configured the two parameters in "yarn-site.xml" file. Many of parameters were tuned according to our tens of experiments, we expected that containers with 2 GB memory and AMs with 4 GB memory could satisfy many data-intensive algorithms. The total usable memory for applications of thecluster was 150 GB. Other configurations such as JVM heap size were also considered. As shown in Table I, daemons of the Hadoop master the Spark master were running on the same nodes, with other nodes being used for storage and computing. RS data were previously uploaded to HDFS. The default block size of HDFS was 128 MB, and the replication factor equaled 3. The heterogeneous cluster was used to simulate the infrastructure used in data centers which consist of many commodity PCs.

B. Experiment Design

We have tracked all the time taken by spark for processing the images to analyze the performance of spark.

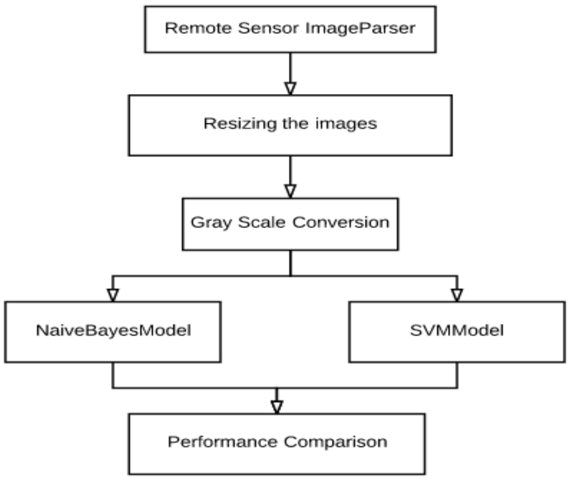
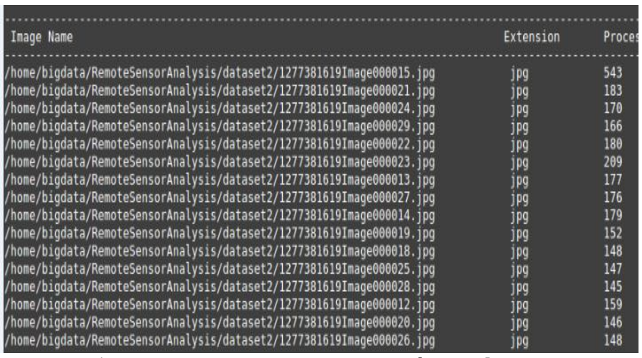


Fig 3: Implementation flow

On average, there were 30 GB of memory used for caching data in the whole procedure. There were at most 40 GB of memory used to cache intermediate data. The results demonstrated that 20%–26% of memory would be used to cache data (30/150GB =0.2,40GB/150GB = 0.26). Nearly 46% of memory would be employed to cache data and execute tasks (70 GB/150 GB = 0.513). This is why many tasks completed in milliseconds as shown in Table III. The load/process footprints showed that even the data-intensive algorithms FVC cannot exceed the entire cluster computing capacity. An interesting phenomenon appeared in the procedure of FVC calculation. From time 0:00 to 1:00, there would be an obvious wave peak, while in the rest of time, there would be a wave trough as shown in Fig. 6. The wave peak indicated that there would be many data caching and shuffle operations occur-ring before actual RS indices calculating stages. The wave trough reflected that the calculating stages would be locally exe-cuted with limited network communication as shown in Fig. 7. The results showed that Spark-based algorithms employ distributed memory to cache data and have constrained networking communication.All of above results show that the strip-oriented parallel programming method is suitable for data-intensive multitasking RS algorithms. Even when the volume of RS data exceeds the size of usable cluster memory, algorithms can obtain perform gains

Efficiency of resizing the images:

We designed a Spark algorithmthat changed pixel digital numbers and count numbers of strips of images. On a Spark standalone model, the algorithm took 150 milliseconds to modify each pixel of 1280X960 strips from 150 GB images, while 30,249 were taken to modify each pixel of 30,24,500 strips from 130 GB images, and 150 s were spent to modify each pixel of 4,56,000 strips from 130 GB images.

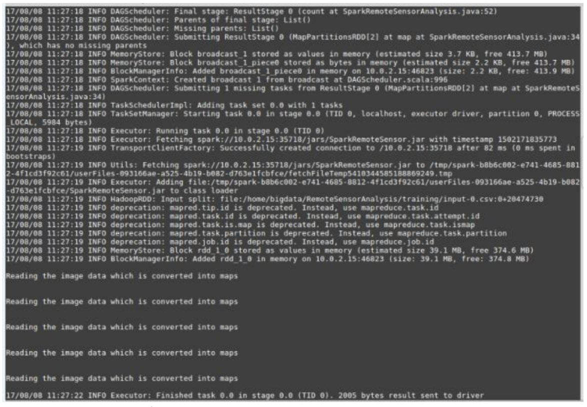


```
---------------------------------------------------------------------------------------------
 Image Name                                                          Extension      Proces
---------------------------------------------------------------------------------------------
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000015.jpg    jpg          543
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000021.jpg    jpg          183
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000024.jpg    jpg          170
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000029.jpg    jpg          166
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000022.jpg    jpg          180
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000023.jpg    jpg          209
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000013.jpg    jpg          177
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000027.jpg    jpg          176
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000014.jpg    jpg          179
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000019.jpg    jpg          152
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000018.jpg    jpg          148
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000025.jpg    jpg          147
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000028.jpg    jpg          145
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000012.jpg    jpg          159
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000020.jpg    jpg          146
/home/bigdata/RemoteSensorAnalysis/dataset2/1277381619Image000026.jpg    jpg          148
```

Fig 4: Image conversion time for each image

We implemented two algorithms, using spark on YARN model. Supposing an image has *N* rows and *M* columns, and each row is small enough to fit in memory. The three filtering primitives are used to obtain three RDDs that represent 0 to *N*-2 rows of the image represented as RDD1, 1 to *N*-1 rows of the image represented as RDD2 and 2 to *N* rows of the image represented as RDD3, respectively. Before applying two joining transformation primitives, two mapping transformation would be applied on RDD2 and RDD3 to change their row numbers to 0 to *N*-2. Rows that have the same keys are then joined togetherusing two joining primitives so that the final RDD can be used to parallel apply Sobel functions on multiple rows need. Although the implementation is not take consideration of edge effect, the example shows that Strip parallelization can tackle the type of parallel. Even when processing a 30 MB image, it took nearly 5s.as shown in above image.

In the below image, 34MB image is processed in SPARK on YARN model just 10secods so Spark on YARN model gives very good resuls as it is an in-memory processing. “WholeFileInputFormat,” frequently used in image processing, introduces a zero partition policy [14]. Therefore, only one mapper instance will be generated to first read all image data and to then process each image in single node. Thus, it is much more inefficient as expected. The “Spark local [200]” model here exhibited a compelling advantage over MapReduce. This model is similar to a multiple-threads model, in which the Spark master and Spark executors are all in one machine. Although it is not reasonable to compare MapReduce with this model, the results show obvious performance gains when it is performed with YARN.



```
17/08/08 11:27:18 INFO DAGScheduler: Final stage: ResultStage 0 (count at SparkRemoteSensorAnalysis.java:52)
17/08/08 11:27:18 INFO DAGScheduler: Parents of final stage: List()
17/08/08 11:27:18 INFO DAGScheduler: Missing parents: List()
17/08/08 11:27:18 INFO DAGScheduler: Submitting ResultStage 0 (MapPartitionsRDD[2] at map at SparkRemoteSensorAnalysis.java:34
), which has no missing parents
17/08/08 11:27:18 INFO MemoryStore: Block broadcast_1 stored as values in memory (estimated size 3.7 KB, free 413.7 MB)
17/08/08 11:27:18 INFO MemoryStore: Block broadcast_1_piece0 stored as bytes in memory (estimated size 2.2 KB, free 413.7 MB)
17/08/08 11:27:18 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on 10.0.2.15:46823 (size: 2.2 KB, free: 413.9 MB)
17/08/08 11:27:18 INFO SparkContext: Created broadcast 1 from broadcast at DAGScheduler.scala:996
17/08/08 11:27:18 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 0 (MapPartitionsRDD[2] at map at SparkRemoteS
ensorAnalysis.java:34)
17/08/08 11:27:18 INFO TaskSchedulerImpl: Adding task set 0.0 with 1 tasks
17/08/08 11:27:18 INFO TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0, localhost, executor driver, partition 0, PROCESS
_LOCAL, 5984 bytes)
17/08/08 11:27:18 INFO Executor: Running task 0.0 in stage 0.0 (TID 0)
17/08/08 11:27:18 INFO Executor: Fetching spark://10.0.2.15:35718/jars/SparkRemoteSensor.jar with timestamp 1502171835773
17/08/08 11:27:19 INFO TransportClientFactory: Successfully created connection to /10.0.2.15:35718 after 82 ms (0 ms spent in
bootstraps)
17/08/08 11:27:19 INFO Utils: Fetching spark://10.0.2.15:35718/jars/SparkRemoteSensor.jar to /tmp/spark-b8b6c002-e741-4685-881
2-4f1cd3f92c61/userFiles-093166ae-a525-4b19-b082-d763e1fcbfce/fetchFileTemp5410344585188869249.tmp
17/08/08 11:27:19 INFO Executor: Adding file:/tmp/spark-b8b6c002-e741-4685-8812-4f1cd3f92c61/userFiles-093166ae-a525-4b19-b082
-d763e1fcbfce/SparkRemoteSensor.jar to class loader
17/08/08 11:27:19 INFO HadoopRDD: Input split: file:/home/bigdata/RemoteSensorAnalysis/training/input-0.csv:0+20474730
17/08/08 11:27:19 INFO deprecation: mapred.tip.id is deprecated. Instead, use mapreduce.task.id
17/08/08 11:27:19 INFO deprecation: mapred.task.id is deprecated. Instead, use mapreduce.task.attempt.id
17/08/08 11:27:19 INFO deprecation: mapred.task.is.map is deprecated. Instead, use mapreduce.task.ismap
17/08/08 11:27:19 INFO deprecation: mapred.task.partition is deprecated. Instead, use mapreduce.task.partition
17/08/08 11:27:19 INFO deprecation: mapred.job.id is deprecated. Instead, use mapreduce.job.id
17/08/08 11:27:19 INFO MemoryStore: Block rdd_1_0 stored as values in memory (estimated size 39.1 MB, free 374.6 MB)
17/08/08 11:27:19 INFO BlockManagerInfo: Added rdd_1_0 in memory on 10.0.2.15:46823 (size: 39.1 MB, free: 374.8 MB)

Reading the image data which is converted into maps


Reading the image data which is converted into maps


Reading the image data which is converted into maps


Reading the image data which is converted into maps


Reading the image data which is converted into maps

17/08/08 11:27:22 INFO Executor: Finished task 0.0 in stage 0.0 (TID 0). 2005 bytes result sent to driver
```

Fig 5: Image processing in YARN

5. DISCUSSION

To coordinate preparing and investigation of expansive scale RS information, we utilized the in-memory computing system known as Apache Spark in this paper. As is known. Notwithstanding, programming against the extraordinary equipment requires countless particular procedures to unequivocally exchange information between the PCI gadgets. Message passing interface (MPI) and CUDA libraries are generally embraced in RS calculations to advance their proficiency. Furthermore, applying the programming strategy is fairly troublesome and requires very much planned booking calculations to adjust workloads amongst CPU and GPU [65]. In this way, the programming model is not appropriate for coordinating preparing and examination of substantial scale RS information. Spark based RS calculations utilize abnormal state programming model and would be effortlessly executed as presented early. Since spark uses RDDs to speak to information apportioned crosswise over circulated hubs, change capacities are equipped for being serialized to every hub and executed locally. In this way, the cost of express information exchanges of extensive information obstructs between handling units is dispensed with. The Spark engine has hidden much booking and blame handler detail as well. Therefore, researchers can focus on algorithmic rationale instead of low-level correspondence control advancements and assignment planning. Hadoop has turned into the most broadly utilized stage for circulated capacity and parallel handling expansive volumes of information. One of its successors, Hadoop YARN, can powerfully rent holders to structures with high unwavering quality and adaptation to non-critical failure limit, and is generally utilized as a part of distributed computing situations. This intense stage for information operations was likewise utilized as a part of this work. Other surely understood GCP likewise adjust administrations situated models for concealing the assortment of processing hubs. Be that as it may, Hadoop YARN use virtualization advancements as opposed to web administrations to this end. YARN compartments are an on-request registering asset and information put away in HDFS that can be parallel handled with MapReduce programming model. In this way, it might be more possible. In any case, the effectiveness of MapReduce-based RS calculations is very corresponded with all around outlined intelligent pictures parcel, which tormented both programming issues and I/O issues.

6. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel spark on hadoop yarn framework based in-memory parallel processing model for integrated processing and analysis of large-scale RS data using Spark on a Hadoop YARN platform.By using strip reflection of RS data and Spark RDDs, all-level RS parallel calculations can be communicated independently of the assortment of picture designs and proficiently executed in a heterogeneous distributed computing condition. RS calculations can be parallelized with mapping, joining, and sifting primitives and P2P-based communicate factors. Our analyses demonstrate that Spark-based RS calculations can productively process vast scale volume of RS data notwithstanding when data volume surpasses memory limit. It was conceivable to process 45 million pixels in 1 s on a little heterogeneous group. Applying complex changes on the greater part a terabyte of RS data finished in a couple of hours. Sharing an expansive volume of data in multitasking RS calculations has restricted effect on the productivity of Spark-based RS calculations. Almost 51.3% memory would be utilized to store data and execute errands, and close to 3% mistakes would happen in data serious calculation stages. Spark-based RS calculations can perform productively, and picture sizes have constrained effect on them. data concentrated unpredictable parallel calculations can be more strong and productive if given more YARN compartments. Contrasted and Spark on Mesos demonstrate, Spark on YARN model can be 1.7× quicker for both joining-escalated and iterative-serious RS calculations. The strip-situated programming empowers RS pictures being mapped to dispersed vectors and, in this manner, parallel RS calculations in light of the model can flawlessly incorporate with Spark MLlib. Besides, they can be effectively transplant to other YARN groups with no alteration for intercluster preparing RS enormous data.. Our future work will focus on optimizing a global cache for Apache Spark and on deep learning from massive RS data for estimation of vehicle accidents prediction based on remote sensor image processing.

References

[1]. J. Famiglietti, A. Cazenave, A. Eicker, J. Reager, M. Rodell, and Velicogna, “Satellites provide the big picture,” Science, vol. 349, pp. 684–685, 2015.

[2] Y. Ma et al., “Remote sensing big data computing: Challenges and opportunities,” Future Gener. Comput. Syst., vol. 51, pp. 47–60, 2015.

[3] H. Xia, H. A. Karimi, and L. Meng, “Parallel implementation of Kaufman’s initialization for clustering large remote sensing images on clouds,” Comput. Environ. Urban Syst., 2014, in progress, Corrected Proof.

[4] J. Li, L. Meng, F. Z. Wang, W. Zhang, and Y. Cai, “A map-reduce-enabled SOLAP cube for large-scale remotely sensed data aggregation,” Comput.Geosci., vol. 70, pp. 110–119, 2014.

[5] L. Mascolo, M. Quartulli, P. Guccione, G. Nico, and I. G. Olaizola, “Distributed mining of large scale remote sensing image archives on pub-lic computing infrastructures,” arXiv preprint arXiv:1501.05286, 2015.

[6] R. Giachetta, “A framework for processing large scale geospatial and remote sensing data in map reduce environment,” Comput. Graph., vol. 49, pp. 37–46, 2015.

[7] Z. Sun, F. Chen, M. Chi, and Y. Zhu, “A spark-based big data platform for massive remote sensing data processing,” in Data Science, New York, NY, USA: Springer, 2015, pp. 120–126.

[8] R. Giachetta and I. Fekete, “A case study of advancing remote sensing image analysis,” ActaCybernetica, vol. 22, pp. 57–79, 2015.
[9] C. Lee, S. D. Gasster, A. Plaza, C.-I. Chang, and B. Huang, “Recent developments in high performance computing for remote sensing: A review,” IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens., vol. 4, no. 3, pp. 508–527, Sep. 2011.
[10] F. Van Den Bergh, K. J. Wessels, S. Miteff, T. L. Van Zyl, D. Gazendam, and A. K. Bachoo, “HiTempo: A platform for time-series analysis of remote-sensing satellite data in a high-performance computing environment,” Int. J. Remote Sens., vol. 33, pp. 4720–4740, 2012.
[11] G. Giuliani, N. Ray, and A. Lehmann, “Grid-enabled spatial data infrastructure for environmental sciences: Challenges and opportunities,” Future Gener. Comput. Syst., vol. 27, pp. 292–303, 2011.
[12] M. Peter and G. Timothy, “The NIST definition of cloud comput-ing,” NIST special publication, 800-145, U.S. Department of Commerce, vol. 9, 2011.
[13] P. Wang, J. Wang, Y. Chen, and G. Ni, “Rapid processing of remote sens-ing images based on cloud computing,” Future Gener. Comput. Syst., vol. 29, pp. 1963–1968, 2013.
[14] M. H. Almeer, “Cloud hadoop MapReduce for remote sensing image analysis,” J. Emerg. Trends Comput. Inf. Sci., vol. 3, pp. 637–644, 2012.
[15] F.-C. Lin, L.-K. Chung, C.-J. Wang, W.-Y. Ku, and T.-Y. Chou, “Storage and processing of massive remote sensing images using a novel cloud computing platform,” GISci. Remote Sens., vol. 50, pp. 322–336, 2013.
M. P. Bajcsy, A. Vandecreme, J. Amelot, P. Nguyen, J. Chalfoun, and Brady, “Terabyte-sized image computations on hadoop cluster plat-forms,” in Proc. IEEE Int. Conf. Big Data, 2013, pp. 729–737.
[16] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, “HaLoop: Efficient iter-ative data processing on large clusters,” Proc. VLDB Endowment, vol. 3, pp. 285–296, 2010.
[17] Y. Zhang, Q. Gao, L. Gao, and C. Wang, “iMapReduce: A distributed computing framework for iterative computation,” J. Grid Comput., vol. 10, pp. 47–68, 2012.
[18] F. Pan, Y. Yue, J. Xiong, and D. Hao, “I/O characterization of big data workloads in data centers,” in Big Data Benchmarks, PerformanceOptimization, and Emerging Hardware, New York, NY, USA: Springer,2014, pp. 85–97.
[19] K. Kambatla and Y. Chen, “The truth about MapReduce performance on SSDs,” in Proc. USENIX 28th Large Install. Syst. Admin. Conf.(LISA’14), 2014, pp. 1–9.
[20] M. Sevilla, I. Nassi, K. Ioannidou, S. Brandt, and C. Maltzahn, “SupMR: Circumventing disk and memory bandwidth bottlenecks for scale-up MapReduce,” in Proc. IEEE Int. Parallel Distrib. Process. Symp.Workshops (IPDPSW’14), 2014, pp. 1505–1514.
[21] C. P. Chen and C.-Y. Zhang, “Data-intensive applications, challenges, techniques and technologies: A survey on big data,” Inf. Sci., vol. 275, pp. 314–347, 2014.
[22] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” in Proc. 2nd USENIXConf. Hot Topics Cloud Comput., 2010, p. 10.
[23] R. Xin, P. Deyhim, A. Ghodsi, X. Meng, and M. Zaharia, “GraySort on apache spark by databricks,” Databricks, Sort in Spark, Nov. 2014.
[24] V. K. Vavilapalliet al., “Apache hadoop YARN: Yet another resource negotiator,” in Proc. 4th Annu. Symp. Cloud Comput., 2013, p. 5.
[25] M. Zahariaet al., “Resilient distributed datasets: A fault-tolerant abstrac-tion for in-memory cluster computing,” in Proc. 9th USENIX Conf. Netw.Syst. Des. Implement., 2012, pp. 2–2.
[26] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google file system,” in ACM SIGOPS Operating System Review. Bolton Landing, New York, USA, 2003, pp. 29–43.
[27] J. Dean and S. Ghemawat, “MapReduce: A flexible data processing tool,” Commun. ACM, vol. 53, pp. 72–77, 2010.
[28] A. Toshniwalet al., “Storm@twitter,” presented at the Proceedings of the2014 ACM SIGMOD International Conference on Management of Data,Snowbird, UT, USA, 2014.
[29] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino, “Apache Tez: A unifying framework for modeling and building data pro-cessing applications,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2015, pp. 1357–1369.
[30] K. Wang et al., “Overcoming hadoop scaling limitations through dis-tributed task execution,” in Proc. IEEE Int. Conf. Cluster Comput.(CLUSTER’15), 2015, pp. 236–245.
[31] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, Learning Spark:Lightning-Fast Big Data Analysis. Sebastopol, CA, USA: O’ReillyMedia, Inc, 2015.
[32] A. Davidson and A. Or, “Optimizing shuffle performance in spark,” Univ. California, Berkeley-Dept. Electr. Eng. Comput. Sci., UC Berkeley, Tech. Rep, 2013, p. 10.
[33] M. Armbrustet al., “Scaling spark in the real world: Performance and usability,” Proc. VLDB Endowment, vol. 8, pp. 1840–1843, 2015.
[34] N. Rana and S. Deshmukh, “Performance improvement in apache spark through shuffling,” Int. J. Sci. Eng. Technol. Res. (IJSETR’15), vol. 4, p. 3, 2015.

[35] T. White, Hadoop: The Definitive Guide. Sebastopol, CA, USA: O’Reilly Media, Inc, 2012.
[36] W. Dai and M. Bassiouni, “An improved task assignment scheme for hadoop running in the clouds,” J. Cloud Comput., vol. 2, pp. 1–16, 2013.

[37] J. Xing and R. Sieber, “Sampling based image splitting in large scale dis-tributed computing of earth observation data,” in Proc. IEEE Int. Geosci.Remote Sens. Symp. (IGARSS’14), 2014, pp. 1409–1412.

[38] A. Ajiet al., “Hadoop GIS: A high performance spatial data warehousing system over MapReduce,” Proc. VLDB Endowment, vol. 6, pp. 1009– 1020, 2013.

[39] N. Ritter et al., “GeoTIFF format specification GeoTIFF revision 1.0,” 2000, pp. 1–95 [Online]. Available: http://www.remotesensing.org/ geotiff/spec/geotiffhome.html.

[40] M. Folk, A. Cheng, and K. Yates, “HDF-5: A file format and I/O library for high performance computing applications,” in Proc. Supercomput., 1999, pp. 5–33.

[41] R. Rew and G. Davis, “NetCDF: An interface for scientific data access,” IEEE Comput. Graph. Appl., vol. 10, no. 4, pp. 76–82, Jul. 1990.

[42] Y. Ma, L. Wang, D. Liu, P. Liu, J. Wang, and J. Tao, “Generic parallel programming for massive remote sensing data processing,” in Proc. IEEEInt. Conf. Cluster Comput. (CLUSTER’12), 2012, pp. 420–428.

[43] T. N. Carlson and D. A. Ripley, “On the relation between NDVI, frac-tional vegetation cover, and leaf area index,” Remote Sens. Environ., vol. 62, pp. 241–252, 1997.

[44] B.-C. Gao, “NDWI—A normalized difference water index for remote sensing of vegetation liquid water from space,” Remote Sens. Environ., vol. 58, pp. 257–266, 1996.

[46] G. H. Ball and D. J. Hall, “ISODATA, a novel method of data analysis and pattern classification,” DTIC Document, 1965.

[47] J. A. Hartigan and M. A. Wong, “Algorithm AS 136: A k-means clustering algorithm,” Appl. Statist., vol. 28, pp. 100–108, 1979.

[48] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler, “Wide area cluster monitoring with ganglia,” in Proc. IEEE Int. Conf. ClusterComput., 2003, pp. 289–298.

[49] Cloudear, Tuning the Cluster for MapReduce v2 (YARN), 2014 [Online]. Available: http://www.cloudera.com/documentation/enterprise/ latest/topics/cdh_ig_yarn_tuning.html.

pp. Z. Jiang et al., “Analysis of NDVI and scaled difference vegetation index retrievals of vegetation fraction,” Remote Sens. Environ., vol. 101, 366–378, 2006.

[50] S.-B. Duan, Z.-L. Li, B.-H. Tang, H. Wu, and R. Tang, “Generation of time-consistent land surface temperature product from MODIS data,” Remote Sens. Environ., vol. 140, pp. 339–349, 2014.

[52] J. Wickham, C. Barnes, M. Nash, and T. Wade, “Combining NLCD and MODIS to create a land cover-albedo database for the continental united states,” Remote Sens. Environ., vol. 170, pp. 143–152, 2015.

[53] C. A. Clark and P. W. Arritt, “Numerical simulations of the effect of soil moisture and vegetation cover on the development of deep convection,” J. Appl. Meteorol., vol. 34, pp. 2029–2045, 1995.

[54] Y. Ding, X. Zheng, T. Jiang, and K. Zhao, “Comparison and validation of long time serial global GEOV1 and regional Australian MODIS fractional vegetation cover products over the Australian continent,” Remote Sens., vol. 7, pp. 5718–5733, 2015.

[55] J. Wang, Y. Zhao, C. Li, L. Yu, D. Liu, and P. Gong, “Mapping global land cover in 2001 and 2010 with spatial–temporal consistency at 250 m resolution,” ISPRS J. Photogramm. Remote Sens., vol. 103, pp. 38–47, 2015.

[56] E. Vermote, S. Kotchenova, and J. Ray, “MODIS surface reflectance user’s guide version 1.3,” in MODIS Land SurfaceReflectance Science Computing Facility, 2011 [Online]. Available:http://www.modissr.ltdri.org/.

[57] Z. Wan, “MODIS land surface temperature products users’ guide,” Inst. Comput. Earth Syst. Sci., Univ. California, Santa Barbara, CA, USA, 2006 [Online]. Available: http://www.icess. ucsb.edu/modis/LstUsrGuide/usrguide. html.

[58] Y. Qu, Q. Liu, S. Liang, L. Wang, N. Liu, and S. Liu, “Direct-estimation algorithm for mapping daily land-surface broadband albedo from MODIS data,” IEEE Trans. Geosci. Remote Sens., vol. 52, no. 2, pp. 907–919, Feb. 2014.

[59] S. Liang et al., “A long-term Global LAnd Surface Satellite (GLASS) data-set for environmental studies,” Int. J. Digital Earth, vol. 6, pp. 5–33, 2013.

[60] A. Strahler et al., “MODIS land cover product: Algorithm theoretical basis document,” University College of London, London, U.K., 1994.

[61] E. Bartholomé and A. Belward, “GLC2000: A new approach to global land cover mapping from earth observation data,” Int. J. Remote Sens., vol. 26, pp. 1959–1977, 2005.

[62] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNetclassifica-tion with deep convolutional neural networks,” in Advances in NeuralInformation Processing Systems, Neural Information Processing Systemconference, 2012, pp. 1097–1105.

[63] X. Menget al., “MLlib: Machine learning in apache spark,” arXiv preprint arXiv:1505.06807, 2015.

[64] R. B. Zadehet al., “linalg: Matrix computations in apache spark,” arXiv preprint arXiv:1509.02256, p. 13, 2015.

[65] L. Fang, M. Wang, D. Li, and J. Pan, “CPU/GPU near real-time prepro-cessing for ZY-3 satellite images: Relative radiometric correction, MTF compensation, and geocorrection,” ISPRS J. Photogramm. Remote Sens., vol. 87, pp. 229–240, 2014.

[66] A. Plaza, Q. Du, Y.-L. Chang, and R. L. King, “High performance com-puting for hyperspectral remote sensing,” IEEE J. Sel. Topics Appl. EarthObserv. Remote Sens., vol. 4, no. 3, pp. 528–544, Sep. 2011.

AUTHOR

V.Sreedevi, B.tech in IT in 2010 first class with distinction form Kakinada Institute of Engg and Technology, JNTU-K, Kakinada. Now pursuing M.Tech in CSE in G. Pulla Reddy Engineering college. Her research interests include parallel processing and big data analysis.

J.SwamiNaik,B.Tech in CSE in 2001 with first class from G.Pulla Reddy Engg.College,Sri Krishna Devaraya University, Anatapurramu. M.Tech in CSE in 2003 with first class from IIT Guwahati.Selected for Ph.D in CSE from JNTUA, Anatapuramu. Has 13.5 years of teaching experience as associate professor, GPREC from July 2008 to till date