Job Scheduling for Optimizing Data Locality in Hadoop Clusters

Aprigio Bezerra*
Universitat Autonoma de Barcelona
Escola d'Enginyeria
Bellaterra, Spain
abezerra@caos.uab.es

Porfídio Hernández
Universitat Autonoma de Barcelona
Escola d'Enginyeria
Bellaterra, Spain
porfidio.hernandez@uab.es

Antonio Espinosa
Universitat Autonoma de Barcelona
Escola d'Enginyeria
Bellaterra, Spain
antoniomiguel.espinosa@uab.es

Juan Carlos Moure
Universitat Autonoma de Barcelona
Escola d'Enginyeria
Bellaterra, Spain
juancarlos.moure@uab.es

ABSTRACT

We describe the use of non-dedicated clusters by a known group of local applications sharing the computational resources with additional bioinformatics MapReduce applications. We have studied how to effectively use the resources shared by both application types during their execution. In order to keep local application execution times unaffected we consider the configuration of a group of parameters of the Hadoop platform. One of the most relevant aspects to consider is the job scheduling policy. Our aim is to allow that tasks from different jobs that handle the same data blocks are grouped to be run on the same node where the blocks are allocated. Experimental results show that our approach outperforms traditional policies.

Keywords

Map-Reduce, job scheduling, bioinformatics, non-dedicated clusters, resouce managment

1. INTRODUCTION

Recent advances in different scientific disciplines are generating vast amounts of data that must be stored and analysed efficiently. In this sense, it is interesting to consider the impact of Next Generation Sequencing technologies that provide whole genome data sets as the result of each experimentation. Large data repositories are usually built on top of computer clusters. In this way, we can use the added computing and storage capacity of all the individual computers. This kind of systems have become the most common source of computing found in the majority of scientific laboratories. At the same time, we can find computer workstation networks that are shared by different users as non-dedicated environments. That is, systems run a well-known group of local tasks usually with a tight schedule. This is the case of computer laboratories devoted to teaching activities in university schools. Such non-dedicated resources can be used for running scientific applications in idle times using tools like Condor [1]. Alternatively, we can use a social contract to define a local application resource usage threshold that should not be compromised. We define our goal as to be able to execute a set of data intensive MapReduce applications using Hadoop [2] platform in a non-dedicated cluster. At the same time, we want to keep local applications unaffected in terms of their execution times. To fulfill this requirement we use control groups [3] to make reservations of needed computational resources. We propose a new Hadoop job scheduler that selects Hadoop tasks from a pending jobs queue. The policy implemented by the scheduler will analyze the available resources of the non-dedicated cluster and the properties of the additional Hadoop tasks to improve resource usage and execution times. In section 2, we describe the background of the work, namely existing job scheduling policies applied in MapReduce systems. Then, in section 3, we describe the applications considered as local and additional bioinformatics load. In Section 4 we present our proposed policy Finally, in section 5 we are showing the experiments done to evaluate the policy presented and, in section 6, the conclusions obtained.

2. RELATED WORK

The increasing volume of existing experimental information to be stored and processed generates problems related to the management, maintenance and fast filtering of large

*Universidade Estadual de Santa Cruz
Departamento de Ciências Exatas e Tecnológicas
Ilhéus, Bahia, Brasil
aalbezerra@uesc.br

EuroMPI '13, September 15 - 18 2013, Madrid, Spain

http://dx.doi.org/10.1145/2488551.2488591

data sets. It is necessary to apply techniques aimed at reducing the use of disk and network resources, trying to improve data locality accesses. For the classical FIFO scheduling algorithm, the problem of locality is not well resolved [4]. For shared clusters, schedulers as Fair-share [5] and Capacity [6] try to combine locality and fairness among other objectives. In many situations we find that those become contradictory goals to achieve.

Zaharia et al [7] proposed the delay scheduling algorithm to increase data locality in MapReduce applications. The algorithm delays tasks scheduling in K time units when a task does not find data on the same node that is free. However, its use is not suitable for long runtime tasks which may cause performance degradation in jobs that have delayed tasks. Furthermore, the value of K must be configured to obtain good results. Setting this value depends on the type of load and the execution environment. Low values cannot guarantee data locality and too high values can cause starvation problems.

Seo et al. [8] proposed a job scheduling policy based on prefetching techniques for clusters with racks. Unfortunately, increasing data locality when assessing racks does not guarantee data locality when evaluating individual computing nodes.

Our proposal is in line with the work of Zhenhua et al. [9], which recommends the need to make global decisions related to the block allocation data files when a set of applications need to share some input file set. The main difference with our work is that they try to solve globally the problem of the allocation using integer linear programming techniques. In our proposed policy tasks are scheduled individually trying to place new tasks that share blocks of input data on the same node. When a node requests a new task, the policy looks for a task that handles the same data block used by the task just completed. The search of tasks to be assigned to a node occurs between different jobs in the queue.

Hadoop is a highly configurable MapReduce [10] framework developed as an Apache project. It is implemented in Java and is composed of two main subsystems: HDFS distributed file system and the Hadoop task running system. Hadoop applies a predefined input data partitioning policy, a scheduling of the program tasks into the different computer nodes, a fault tolerance management policy, and a transparent communication pattern between the different tasks of the application. With this model, programmers without much experience in distributed environments can effectively use a potentially large system.

In summary, we need to adapt policies and parameters of Hadoop when running static workloads on top of middle-small sized non-dedicated clusters. We will analyse the impact of the framework parameter adjustments on the performance of the applications to look for efficiency without affecting local applications.

Other existing projects are adapting MapReduce frameworks to use shared computing resources. Projects like Moon [11] have carried out a similar work of adjusting a MapReduce framework to an opportunistic environment. Also, Adapt project [12]] proposes the use of placement strategies in order to improve the performance of Map-Reduce applications where computing nodes enter and leave the system at any time. Purlieus [13] provides a Hadoop cluster made of virtual machines as a cloud data center.

3. RUNNING HADOOP JOBS IN A NON-DEDICATED CLUSTER

Our experimentation analyzes the sharing of computational resources among two different kind of applications. First of all, we describe the characterization of a group of local applications. Then, we describe the additional parallel applications we are going to execute concurrently. Finally, we introduce a new job scheduling policy to improve the resource utilization of our non-dedicated cluster. Additionally, we need to ensure that local user applications are not affected during their execution. To do that, we are reserving the resources they need with the use of resource pools named containers.

We propose the use of Linux container implementation. The control groups, cgroups, allow us to reserve resources on the cluster nodes. The resources of each node are organised in a subsystems hierarchy where each cgroup defines a specific percentage of use of a subsystem. From there, we can assign applications, users and processes to each cgroup to define how any combination of cluster resources will be shared.

3.1 Local user activity

Local user activity is modeled by a parameterized benchmark suite. In this way, each program defines a percentage of the actual resource consumption: CPU, memory, disk, and network bandwidth. We use these parameters to represent typical Best-Effort applications that run locally in our system. We have profiled our teaching laboratories during some weeks to have a list of realistic values for comparing the benchmark execution. Table 1 shows a characterization of local applications we considered for our work. In this table we define five different profiles of use of local resources, being a selection of Network Attached Storage (NAS) serial and parallel benchmarks.

3.2 Bioinformatics application

Original Mapping and Assembly with Quality (MAQ) algorithm [15] is designed to align short read DNA sequences to a reference genome. To do that, it builds a hash table to store and index the short read input set. MAQ alignment algorithm is based on the observation that a full length alignment of an n bp long read with at most k differences must contain at least one exact alignment of $\frac{n}{(k+1)}$ consecutive bases [16]. That is, for a 30 base pairs (bp) read to be mapped to a reference sequence with only one mismatch, there must be at least 15 consecutive bases that match exactly independently from the actual position of the mismatch.

MAQ builds six different hash tables to index the first 28 bp of the reads to ensure that alignments with up to two mismatches are hit. These six tables correspond to six non-contiguous templates like 11110000, 00001111, 11000011, 00111100, 11001100 and 00110011 if reads were 8 characters long.

After the read hashes are built, the reference is scanned on forward and reverse strands; each reference sequence scanned is looked up at the hashes. If a hit is found to be a read, MAQ calculates a quality score for the match as the probability that the alignment is wrong. For this, it uses a quality score (Q_s) equation 1.

Where q_1 is the sum of quality values of mismatches of

Profile	NAS	%CPU	%MEM	DISK read	DISK write	NET recv	NET sent
A	EP.A (serial)	99	0.2	26	7	1.32	2.06
B	MG.B (serial)	99	42	37	23	3.84	3.42
C	IS.C (serial)	20	95	23000	12000	0.77	0.72
D	CG.A (parallel)	40	55	54	21	1890	1845
E	FT.A (parallel)	63	61	67	17	1025	965

Table 1: Characterization of the local load using NAS applications. Disk metrics are in blocks/sec and network in Bytes/sec

the best hit, q_2 is the corresponding sum for the second best hit, n_2 is the number of hits with the same number of mismatches as the best hit, k' is the minimum number of mismatches in the 28bp seed, and q is the average value of base quality in the 28bp seed. Other examples of similar spaced seed indexing algorithms like MAQ are SOAP [17] and RMAP [18].

$$Q_s = \min\{ q_2 - q_1 - 4.343\ log\ n_2, \quad 4 + (3 - k')(q - 14) - 4.343\ log\ p_1(3 - k', 28) \} \tag{1}$$

3.2.1 MapReduce MAQ implementation

Bioinformatics applications that define workflows processing large data sets can benefit from the use of MapReduce computing platforms [19] as they provide a useful way of aggregating disk capacity of each machine and then run the needed processing locally on each node.

MapReduce applications can be classified depending on the volume of the data processed in the different stages of the execution. We will have Map input heavy data intensive applications when we have a large collection of data, then Map output and Reduce work with a smaller amount of data. Map and Reduce input heavy applications are those with large Map input and output to be processed at the Reduce phase, where output data of smaller size is produced. Lastly, we can have Reduce input heavy data applications where the Map outputs are much bigger than the rest of the data sets of the application.

Mapreduce MAQ (MrMAQ) [14] is a Reduce input heavy application programmed in Java using Hadoop open source MapReduce implementation. The basic principles of the application are the same as MAQ. That is, to find matches of a list of short reads with a provided genome reference by using a seed and extend algorithm.

The design of MrMAQ is based on previous MapReduce bioinformatics applications like Cloudburst [20], Crossbow [21], and MrsRF [22]. In our particular implementation, Map tasks read both reference and read files to generate 28 bp seeds from both. Reducers receive key-value pairs with matching read and reference sequences, known as hits. They extend the alignment for all hits found, calculate their alignment qualities and return a list of all short reads, their alignment positions in the reference and their alignment qualities.

Map phase: process short read files and genome reference files. Map tasks use their output as the hash tables of the original MAQ. That is, they create a new key from each read storing the result of applying each of the six templates. Then, Maps scan the reference sequence generating a new key for each reference subsequence. The result of this Map phase will be a list of key-value seeds coming from both types of input, as expressed in Figure 1.

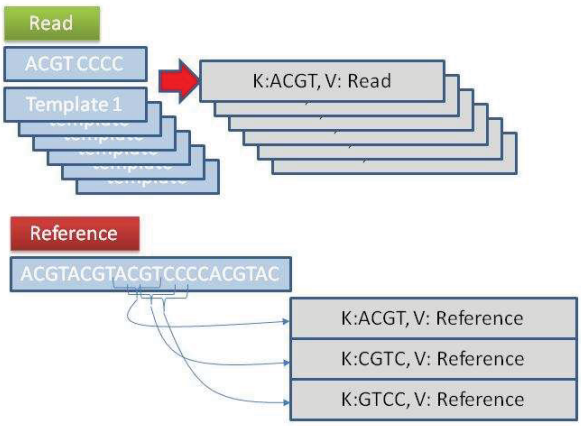

Figure 1: Map Phase

Reduce phase: the alignment extension. For every key that a Reduce task receives, it also receives a set of all reads and references that have the same seed. For each of these matches, the Reduce task will extend the alignment to the rest of the read and calculate the sum of qualities of mismatched bases q over the whole length of the read, extending out form the 28-bp seed without gaps. Figure 2 shows the process of extending the alignment of the hit found. Then it outputs the best quality alignments found for every reference position.

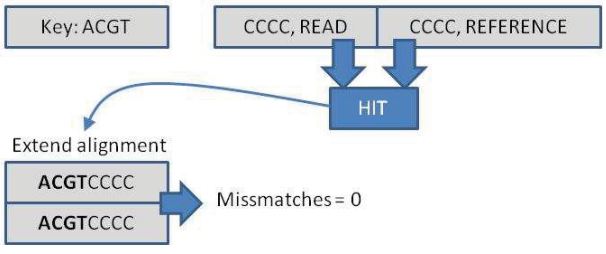

Figure 2: Reduce Phase

4. PROPOSED POLICY

Our job scheduling policy seeks to take advantage of shared input file among different jobs to improve data locality. It was developed for data-intensive applications, such as for

example bioinformatics applications like read-mapping applications. In these applications the input consists of two data files: read and reference files. Many different jobs can share the same reference file even if they use different read files. Our main goal is to allow tasks that share the same input block can be scheduled sequentially and in the same computation node even being tasks of different jobs. This not only ensures a higher data locality but also take advantage of the cache Hadoop Distribuited File System. When a node requests a new task, the policy looks for a task that handles the same data block used by the task just completed. The search of tasks to be assigned to a node occurs between different jobs in the queue. The policy also ensures the possibility that a task could also be sent to a node that has no local data to be processed, if no other tasks are waiting.

The scheduling policy implemented follows a simple mechanism shown in in the algorithm in figure 3. When a computing node task management daemon (TaskTracker) has a free slot to begin a task, sends a heartbeat to master daemon (JobTracker), reporting its status. Then, JobTracker searches among the tasks that are waiting to be executed one that handles the same data block already used on the same node. If it finds this task, it will be send to the node, keeping the data locality, lines 4-9. Otherwise, it seeks a task that processes a block saved in the same node that requested a new task, lines 10-13. If JobTracker does not find any task, then it will send the first pending task to be performed, lines 15-18.

```
1 for each event scheduling of taskTracker:
2 while jobs_in_the_queue do
3 //blockID has the last block ID processed by the taskTracker
4  if (blockID != NULL)
5        if (obtainSharedBlockTask(taskTracker, blockID) != NULL)
6             select(task);
7             break;
8         end_if
9  end_if
10 if (obtainNewLocalTask(taskTracker) != NULL)
11       select(task);
12        break;
13 end_if
14 else
15      if(obtainNewNoLocalTask(taskTracker) != NULL)
16            select(task);
17            break;
18       end_if
19  end_else
20 end
```

Figure 3: Scheduling policy for tasks using the same input data blocks

5. RESULTS

To evaluate the performance of local and Hadoop workload processing, we have used a computer cluster consisting of 9 nodes interconnected by a 100 Mbit network. Nodes are dual processor Intel Pentium 3,4 GHz, with 1 GB of memory and 44 GB of disk for each node. Hadoop version used was 0.20.2 with Oracle Java JDK 1.6.0-16. All nodes where running Linux version 2.6.31. Our non-dedicated cluster runs

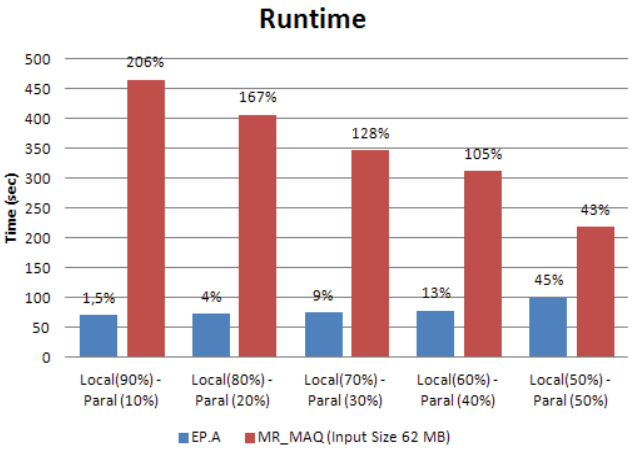

Figure 4: Execution of local and parallel load using cgroup resource reservation

well-known controlled local load. This load varies over time, allocating a few hours to each local application. We are using a list of cgroups to define different resource usage configurations to be used by each node of the system during the normal day activities. Then, we define a concurrent mix of local and additional Hadoop load with the objective of assuring that the local applications are not affected. Figure 4 shows the execution times of the concurrent application mix when decreasing the amount of resources reserved for the local load. From there, for the rest of the analysis, we select those levels that provide close to original execution times reserving the least amount of local resources.

Multiple instances of Hadoop applications contend for access to a common set of input files. As these applications are launched, all their map and reduce tasks are available in the Hadoop task queue. Those tasks from any application that request the same input data blocks can be co-scheduled to be executed together. In our case, Hadoop jobs are instances of read-mapping MrMAQ application. We create a workgroup by considering all instances that share the same genome reference file. We have developed a new scheduling policy to seek affinity between those tasks in a workgroup that request access to the same input blocks. By this, we want to improve the co-cheduling of tasks that share resources like common input files or need to access the same resource like CPU. A workgroup is treated as a new job entity to be executed by Hadoop.

Our work environment is based on the principle that all applications that share files already form a group before their execution. This assumption is not realistic in most cases. In our cluster, the different instances of the read mapping application arrive at known times and, therefore, can be launched in specific times. Scheduling system also needs to consider a limited job queue size and a job admission module that evaluates the number and size of workgroups that are allowed to be in the queue and launched concurrently. In this way, we can avoid long waiting times for tasks in the execution queue.

In our case, each job submitted to the cluster had an input size of 1GB, 2GB and 4 GB. Input files were automatically divided by Hadoop into blocks of 64 MB that were

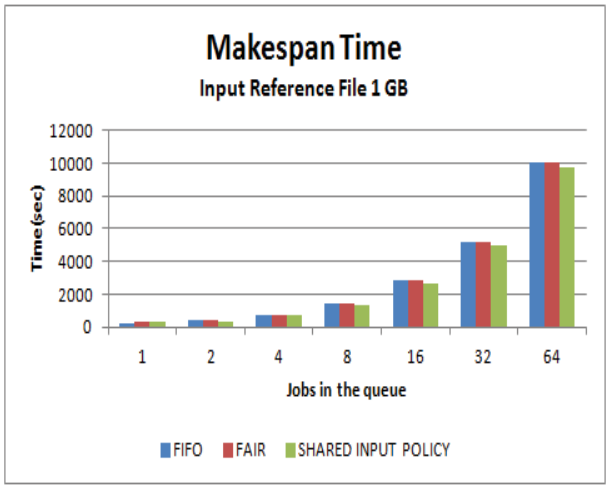

Figure 5: Makespan times of a workgroup execution - 1GB input reference file

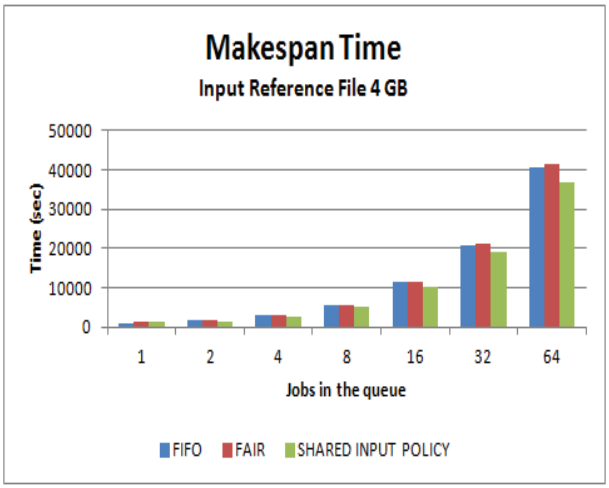

Figure 7: Makespan times of a workgroup execution - 4GB input reference file

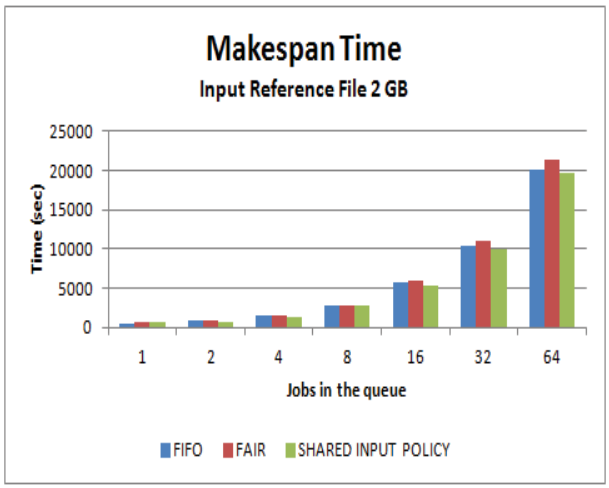

Figure 6: Makespan times of a workgroup execution - 2GB input reference file

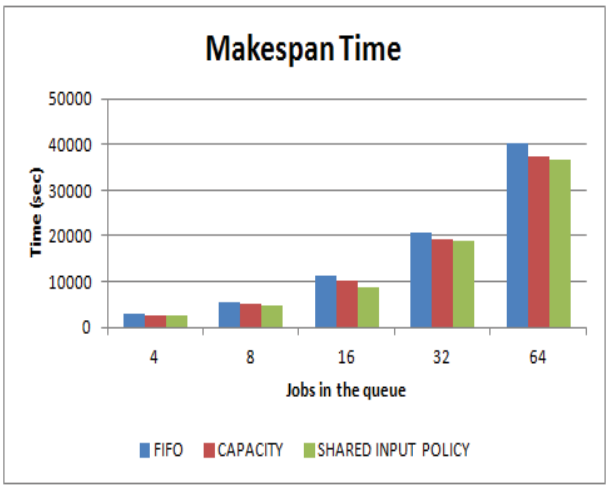

Figure 8: Makespan times of 4 workgroup execution

distributed between the nodes running HDFS. To evaluate the effects of the block distribution in data locality we did not use block replication. All jobs submitted to the cluster were composed of 14 reducers. Figures 5, 6 and 7 shows the execution times of a workgroup with each job sharing the same genome reference file. We evaluate the execution time increasing the amount of jobs queued from 1 to 64. Makespan time was used as a metric for evaluating the scheduling policy. That is, the total time since the first job was submitted to the cluster until the end of the last execution of job queued. Scheduling policies compared were FIFO, Fair scheduler and the proposed policy. Fair policy uses a pool of jobs, while the rest use a job queue. Results show that the policy proposed improves the average makespan time in 7.8% when compared to FIFO and 8.3% against Fair Scheduler.

In the second set of experiments we used four different reference files. That is, we were executing four workgroups concurrently to compare the execution times using FIFO, Capacity, and the proposed policy. For the case of Capacity scheduler, we defined four queues, one for each workgroup. Then, jobs submitted to the cluster were assigned to the corresponding workgroup queue. Cluster resources like available slots were distributed evenly among all queues. The other policies were using a single job queue. Figure 8 shows makespan times for an increasing amount of jobs belonging to the four workgroups. The number of jobs belonging to each workgroup is divided equally among the number of jobs issued. Results show that the policy proposed improves the average makespan time in 7.3% when compared to FIFO and 3.2% when compared to Capacity scheduler. Capacity scheduler creates different jobs queues for each reference file. The jobs of each queue are executed sequentially allowing Hadoop take advantage of the HDFS cache. Still, our policy improves outcomes because it is able to schedule sequentially tasks of different jobs that share the same input data.

6. CONCLUSIONS AND FUTURE WORK

We have analysed the adaptation of Hadoop MapReduce framework to run various instances of bioinformatic applications in a non-dedicated computer system. We also have presented a new scheduling policy that defines workgroups of applications so that they are coscheduled together. The execution of these Hadoop bioinformatic applications must be adapted to use existing computing resources so that local applications are not affected. We propose the use of cgroups to make adequate resource reservations during the daily use of our non-dedicated computer system. We have compared our policy with those available in Hadoop standard installation to show the benefits of the new policy.

Next steps in the research will consider the dynamic definition of local resources and the impact of local resource occupation to the Hadoop application workgroups. For that case, the scheduler will need to dynamically update its choice of applications to consider tasks that fit better the available resources.

7. ACKNOWLEDGEMENTS

This work has been supported by projects number TIN2007-64974 and TIN2011- 28689 of Spanish Ministerio de Ciencia y Tecnologia (MICINN).

8. REFERENCES

[1] D. Thain, T. Tanembaum, M. Livny, *Distributed Computing in Practice: The Condor Experience*, Concurrency and Computation: Practice and Experience **17** (2005) 323–356.

[2] T. White, *Hadoop. The Definitive Guide*, Second edition. O'Reilly, Sebastopol, 2011.

[3] G. Banga, P. Druschel, J.C. Mogul, *Resource containers: A new facility for resource management in server systems*, Proceedings of OSDI 1999 (1999) 45–58.

[4] B. Thirumala Rao, L. S. S. Reddy, *Survey on Improved Scheduling in Hadoop MapReduce in Cloud Environments*, International Journal of Computer Applications 34 (2011) 29–33.

[5] Fair Scheduler, Tech. rep., Retrieved: February, 2012. `http://hadoop.apache.org/common/docs/r0.20.2/fair_scheduler.html`

[6] Capacity Scheduler, Tech. rep., Retrieved: February, 2012. `http://hadoop.apache.org/common/docs/r0.20.2/capacity_scheduler.html`

[7] M. Zaharia, D. Borthakur, J.S. Sarma, K. Elmeleegy, S. Shenker, I. Stoica, *Delay scheduling: a simple technique for achieving fairness in cluster locality and scheduling*, Proc. of the 5th ECCS (2010).

[8] S. Seo, I. Jang, K. Woo, I. Kim, J. S. Kim, S. Maeng, *HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment*, IEEE International Conference on Cluster Computing and Workshops (2009) 1-8.

[9] Z. Guo, G. Fox, M. Zhou, *Investigation of Data Locality in mapReduce*, Tech. Report, Indiana University 11/27/2011.

[10] J. Dean, S. Gemawat and J. A. Wheeler, *Map-reduce: simplified data processing on large clusters.*, ACM Communications **51** (2008) 107–113.

[11] H. Lin, J. Archuleta, W. Feng, M. Gardner, Z. Zhang, *MOON: map-reduce On Opportunistic eNvironments*, Proc. of the 19th ACM HPCD 2010.

[12] H. Jin, X. Yang, X. Sun, I. Raicu, *ADAPT: Availability-aware map-reduce Data Placement for Non-Dedicated Distributed Computing*, Proc. of ICDCS 2012.

[13] B. Palanisamy, A. Singh, L. Liu, B. Jain, *Purleius: Locality-aware Resource Allocation for map-reduce in a Cloud*, Proc. of ACM/IEEE Conf. on Supercomputing 2011.

[14] A. Espinosa, P. Hernandez, J.C. Moure, J. Protasio, A. Ripoll, *Analysis and improvement of map reduce data distribution in read mapping applications*, Journal of Supercomputing. To appear. DOI 10.1007/S11227-012-0792-8

[15] H. Li, J. Ruan, R. Durbin, *Mapping short DNA sequencing reads and calling variants using mapping quality scores*, Genome Research 2008.

[16] R. A. Baeza-yates, C. H. Perleberg, *Fast and Practical Approximate String Matching.*, In Combinatorial Pattern Matching, Third Annual Symposium **51** (1992) 185–192.

[17] R. Li, Y. Li, K. Kristiansen, J. Wang, *SOAP: short oligonucleotide alignment program.*, Bioinformatics, 24, 5 **51** (2008) 713–714.

[18] A. Smith, Z. Xuan, M. Zhang, *Using quality scores and longer reads improves accuracy of Solexa read mapping.*, BMC Bioinformatics, 9, 1 **51** (2008) 128.

[19] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, M. A. DePristo, *SThe Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data.*, Genome Research, 20, 9 **51** (2010) 1297–1303.

[20] M. C. Schatz, *CloudBurst: highly sensitive read mapping with MapReduce.*, Bioinformatics, 25, 11 **51** (2009) 1363–1369.

[21] B. Langmead, M. Schatz, J. Lin, M. Pop, S. Salzberg, *Searching for SNPs with cloud computing.*, Genome Biology, 10, 11 **51** (2009) R134.

[22] S. Matthews, T. Williams, *MrsRF: an efficient MapReduce algorithm for analyzing large collections of evolutionary trees.*, BMC Bioinformatics, 11 **51** (2010) 1–9.