Spark on Entropy: A Reliable & Efficient Scheduler For Low-latency Parallel Jobs in Heterogeneous Cloud

Huankai Chen
Future Computing Group
School of Computing
University of Kent
Canterbury, Kent, UK
Email: HC269@kent.ac.uk

Frank Z Wang
Professor of Future Computing
School of Computing
University of Kent
Canterbury, Kent, UK
Email: F.Z.WANG@kent.ac.uk

***Abstract*—In heterogeneous cloud, the provision of quality of service (QoS) guarantees for on-line parallel analysis jobs is much more challenging than off-line ones, mainly due to the many involved parameters, unstable resource performance, various job pattern and dynamic query workload. In this paper we propose an entropy-based scheduling strategy for running the on-line parallel analysis as a service more reliable and efficient, and implement the proposed idea in Spark.**

Entropy, as a measure of the degree of disorder in a system, is an indicator of a system's tendency to progress out of order and into a chaotic condition, and it can thus serve to measure a cloud resource's reliability for jobs scheduling. The key idea of our Entropy Scheduler is to construct the new resource entropy metric and schedule tasks according to the resources ranking with the help of the new metric so as to provide QoS guarantees for on-line Spark analysis jobs. Experiments demonstrate that our approach significantly reduces the average query response time by 15% - 20% and standard deviation by 30% - 45% compare with the native Fair Scheduler in Spark.

Keywords*—*Entropy Scheduler, Reliable, Low-latency, Spark, Jobs Scheduling, Heterogeneous Cloud, Cloud Computing

I. INTRODUCTION

As Cloud computing continues to evolve, an increasing number of companies are exploiting the Spark framework [8] as an online web analysis platform with sub-second query latency on the cloud. The analysis applications range in functionality, complexity, resource needs and service delivery deadlines. This diversity creates competing requirements for program design, job scheduling and resource management policies on cloud environments. In spite of different user objectives one goal is common: to provide low-latency and reliable web service to the service consumers. However, scheduling low-latency parallel jobs under heterogeneous cloud poses a number of challenges.

Scheduling is an NP-complete problem, the complexity of which increase substantially in heterogeneous cloud environment. Developers who construct scheduling system must cope with the world's natural tendency to disorder. In Spark, jobs are scheduled on a set of cloud resources that are local active (in the sense that each resource was determined to be assigned jobs based its own state and the state of the environment and its productivity are affected by the amount of jobs that assigned to it), and corporately structured. We want resource local activity to yield coherent global schedule system order. However, widespread experience warns us that optimizing systems that exhibit both local activity [17] and global order are not easy. The experience that anything that can go wrong will go wrong and at the worst possible moment is summarized informally as "Murphy's Law" [18]. Scheduling systems are not immune to Murphy. In cloud scheduling system, after an enough power strikes one of the resources, which leads to its productivity reduced or collapsed, the whole system collapsed. In the real world scenario, such resource productivity reduced or collapsed may cause by hardware/software failures, resource CPU overload, resource over- or under-provisioning, or application misbehaviors. Thus, the system is failed to execute and complete jobs as expected.

At the root of the ubiquity of disordering tendencies is the Second Law of Thermodynamics, "Energy spontaneously tends to flow only from being concentrated in one place to becoming diffused or dispersed and spread out" [19]. In cloud scheduling system, allocating the "right" type of resources to jobs may overcome the Second Law "spontaneous tendency" and lead to increasing the system's order. However, the way to decide which resource is "right" for the jobs is critical. Especially when resources are local activity, which is the origin of complexity [17], the scheduling system become more complex under heterogeneous cloud environment.

In the literature, a lot of scheduling algorithms were proposed in the past. Braun et al [20] have studied the relative performance of eleven heuristic algorithms for task scheduling such as First Come First Served (FCFS), Min-Min, Max-Min, Genetic Algorithm (GA), etc. A family of 14 scheduling heuristics for concurrently executing BoTs in cloud environments are proposed recently [21]. Most of the past works are the focus in shortening project's completion time and enhancing the system throughput without considering the reliability factor. A theoretical scheduling paper address the reliability problem of scheduling system by analyzing the entropy [4]. However, it calculates the entropy base on jobs instead of resources. We argue that such technique has limited capacity for modeling the complex cloud system that

resources' performance are highly dynamic and nonlinear.

Both efficiency and reliability are particularly important for processing online production queries when a given set of Spark jobs needs to be executed simultaneously on the cloud. The reduced efficiency and reliability of the global cloud system is a direct consequence of the disorder caused as a result of the local active resources and the difficulty in managing these resources. Thus, the resulting scheduling problem is also an entropy-optimization problem: how to schedule jobs to the "right" resources with the help of local resource entropy to keep the global cloud system working in "order" state. The fundamental claim of this paper is to solve the above cloud scheduling problem based on Entropy Theory.

The remainder of this paper is organized as follows: Section 2 describes the Spark analysis as a service framework and the related scheduling problem; Section 3 covers Entropy Scheduler's architecture and implementation details; Section 4 presents empirical results from benchmarking Entropy Scheduler with the native Spark FAIR scheduler; Section 5 contains concluding remarks, lessons learned and possible future research direction.

II. SCHEDULING CHALLENGE IN SPARK ANALYSIS AS A SERVICE

This section provides a brief overview of Spark and discusses the current scheduling challenge for deploying Spark analysis as a service.

A. Spark

Spurred by demand for lower-latency distributed data analysis, efforts in research and industry alike have produced engines such as MapReduce, Hive, Dremel, Impala and Spark that run cloud analysis jobs across thousands of resources in short time, as shown in Figure 1. Spark is part of the Apache Software Foundation and claims speedups up to 100x faster than Hadoop MapReduce in-memory, or 10x faster on disk. The ability to bring response times into sub-second range has enabled powerful new application development - Spark Analysis as a service. In such case, user-facing services will be able to run sophisticated parallel computation, such as language translation, voice reorganization, highly search personalizations and context recommendation, on a per-query basis.

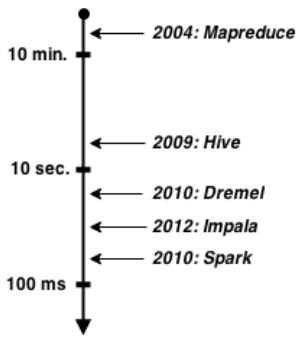

Fig. 1. Cloud engines can run parallel analysis jobs with ever lower latency.

B. SAAAS:Spark analysis as a service

SAAAS provides spark analysis query requests and response over HTTP web service, which support multi-threaded querying. Figure 2 illustrates the flow of sending an HTTP requests. Spark Web Server allocates a thread to route the HTTP request to a specify Spark analysis job. Jobs are then processed with a long run global Spark Context and scheduled by the Spark Master to run on the predefined amount of Spark Workers.

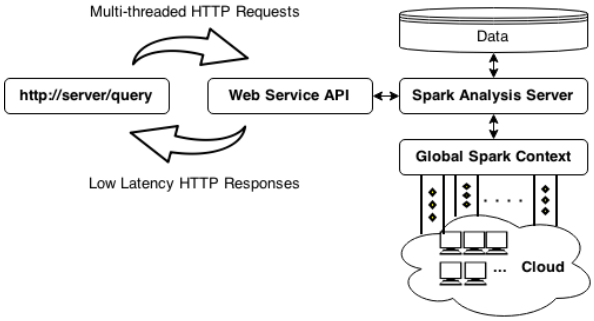

Fig. 2. SAAAS:Spark analysis as a service.

C. Scheduling in Spark : Fair Scheduler with Random Resource Sorting

The Spark Context supports multi-threading and offers FIFO and FAIR scheduling options for concurrent queries. Typically, the FAIR scheduler is used for processing multiple parallel jobs simultaneously to minimize overall latency. The purpose of FAIR scheduler is to assign resources to queries such that all queries get an equal share of resources over time on average. By default, the scheduler bases fairness decisions only on the number of resource cores and memory and assign jobs to the resource offers with random sorting. The FAIR scheduler does not consider the core speed and CPU utilization of the resource, which have a direct impact on the completion time of the jobs. Thus, it is hard to guarantee the QoS for the on-line query. If the scheduling strategy cannot provide an optimal way to guarantee the QoS, it will be difficult to popularize the web service.

D. Problem Statement

Scheduling low-latency parallel analysis jobs on the heterogeneous cloud is a challenging, multifaceted problem. Although motivated by and designed for the Cloud, Spark engines have not yet addressed the problem of resource scheduling for high concurrent jobs on the heterogeneous cloud. Spark's performance is closely tied to its job scheduler, which implicitly assume that cloud resources are homogeneous and resource's performance is never changed during run-time, and use these assumptions to decide how to allocate jobs to resources. In practice, the homogeneity assumptions do not always hold, and the performance of the resource is highly dynamic. Although the current scheduler works well in homogeneous environment, we show it can lead to severe performance degradation when its underlying assumptions are broken: the performance of resources with potentially uncontrollable variance and the server collapse when meeting high concurrent requests. Furthermore,

we expect heterogeneous environments to become the common case as organizations often use multiple generations of hardware for building their private cloud.

In this paper, we address the question of how to efficiently and reliably schedule query jobs on Spark analysis cloud. In particular, we focus on the following problems: Resource allocation to a given job according to resource core speed, CPU utilization and Entropy Level rather than treating the resource as a homogeneous array of cores. The goal is to find a deployment that improve the service performance and QOS guarantees.

III. ENTROPY SCHEDULER : A MORE RELIABLE AND EFFICIENT SOLUTION

Optimized resource management and scheduling must take into consideration: (1) the characteristics and activity of the individual resource, and (2) the reliability of information gain from the resource. In this section, We propose to use resource activity vector to characterize the behaviour of individual resource and calculate the resource entropy level based on that vector to measure how reliable such information is. We then present a resource ranking algorithm that uses the resource activity vector and entropy level together to guide the jobs scheduling to improve system's performance and stability.

A. Entropy Theory

Entropy is an important statistical quantity that measures the disorder degree and the amount of wasted energy in the transformation from one state to another in a system [12]. Although the concept of entropy was originally a thermodynamic construct, it has been adapted in other fields of study, including information theory, production planning, resource management, computer modeling, and simulation [3] [5] [13] [14] [15] [16]. We will use this measure to quantify the reliability degree associated with the information gain from the resources under Spark scheduling strategy on the heterogeneous cloud. We introduce the Entropy measure in general content. Given a dynamic system X of finite mutually exclusive state variable set $S = s_1, s_2, s_3, \ldots, s_n$ with probabilities $p_1, p_2, p_3, \ldots, p_n$ respectively, entropy $H(X)$ is defined as:

$$H(X) = -\sum_{i=1}^{n} p_i * \log p_i \tag{1}$$

B. Resource Activity Vector & Entropy Level

Good jobs scheduling requires awareness of the resource characteristics. In the heterogeneous cloud, the system's performance have become more sensitive to the resources they have at hand, and poor scheduling can lead to performance degradation. However, the native Spark Fair scheduler only considers the static characteristics of the resource, such as number of available cores, while ignores the dynamic characteristics like core performance. In such situation, jobs are 'unfairly' scheduled on the cores with different performance, which highly impact on the completion time of the jobs and predictable of system performance.

In order to capture the relevant dynamic core performance characteristic, we introduce resource activity vector (RAV) and resource entropy level (REL). In the current implementation, we concentrate on the most important part of resource information, CPU utilization, which represents how efficiently the operator thread uses the CPU throughout the jobs execution. This is highly relevant for making scheduling decision as it is directly related to the core's performance during run-time. To obtain the RAV values, we run a resource monitor on each worker node. The resource monitor captures the worker's CPU utilization and updates the RAV with the CPU utilization difference every second. The REL is updated according to the algorithm 1 on every heartbeat interval. Then the worker node sent the heartbeat to the master node with its current CPU utilization value and entropy level for making jobs scheduling decision.

Algorithm 1 Calculate Resource Entropy Level

Require: $RAV \leftarrow ResourceActivityVector$
Ensure: $RAV.size \geq 0$
  $Avg \leftarrow AverageChangedOfCPUUtilization$
  $REL \leftarrow ResourceEntropyLevel$
  $Count1 = 0$
  $Count0 = 0$
  for all $Value$ in RAV **do**
    if $Value \geq Avg$ **then**
      $Count1 = Count1 + 1$
    else
      $Count0 = Count0 + 1$
    end if
  end for
  $P1 = Count1/RAV.size$
  $P0 = Count0/RAV.size$
  $REL = -(P1 * \log_2 P1 + P0 * \log_2 P0)$

C. Schedule Jobs by the Ranking of Resource

Spark assumes that all the resource are homogeneous and randomly assign cores to jobs under Fair Scheduler. However, even in the homogeneous cloud, resources with 'homogeneous' setting will always running under 'heterogeneous' performance during run-time. Especially in heterogeneous cloud, such assumption will easily result in poor jobs completion time and overall unstable cloud performance due to the following reason:

- Job completion time is decided by the completion time of its slowest task.
- Random cores allocation will increase the chance of allocating cores with different performances to tasks inside a single job.
- Cores are not released for scheduling other jobs until the current running job is completed. When a job is waiting for its slowest task to be completed, the computing power of other cores with completed tasks are wasted.
- Monitoring and re-scheduling slow tasks (performs the speculative execution of tasks) is expensive.

In our proposed Entropy Scheduler, instead of randomly pickup resources, we first calculate the performance ranking of all offered resources (Algorithm 2), and then schedule tasks inside a job according to the ranking. Tasks are scheduled with similar ranking resource so as to improve overall jobs completion time and reliability of cloud performance.

Algorithm 2 Calculate Resource Performance Ranking

Require: $RCU \leftarrow ResourceCPUUtilization$
Require: $REL \leftarrow ResourceEntropyLevel$
Require: $NumCores \leftarrow NumberofAvailableCores$
Require: $CpuSpeed \leftarrow CPUclockspeed$
$Rank \leftarrow Resourceperformanceranking$
$Rank = NumCores * CpuSpeed * (1-RCU) * (1-REL)$

IV. EMPIRICAL EVALUATION OF ENTROPY SCHEDULER

A. Experimental Platform

All experiments leverage a heterogeneous 3 node production cluster of HP ProLiant DL360 G5 blades. The nodes specifications are shown on Table I. Furthermore, each node has Ubuntu Server 14.04.2 LTS, Spark 1.3.0, sbt 0.13.5. Each Spark worker uses all the available cores and 8g of memory.

Section II-B describes SAAAS's ability to process concurrent queries with multi-threading since many users will use the analysis service concurrently. Users will request a different type of queries concurrently as well, but for simplicity, these experiments only benchmark the same query to calculate Pi with 4 cores. We use Apache Bench to load testing the query on the Spark Web Server under different schedulers (Entropy Scheduler and native Fair Scheduler). The load testing will spawn a number of threads which continuously execute the same query. Each thread remains loaded and continues processing queries until all threads have finished, and the query response time of all requests from every thread will be used for performance comparison.

TABLE I. EXPERIMENTAL PLATFORM:RESOURCE SPECIFICATION

Specification	Node 1	Node 2	Node 3
Spark Role	Master&Worker	Worker	Worker
CPU	Xeon 3Ghz x 2	Xeon 2.8Ghz x 2	Xeon 1.8Ghz
Number of Cores	8	8	4
RAM	16GB	12GB	12GB

B. Experiment 1 : Performance under Different Concurrent Level of HTTP Request Workload

This experiment is used to verify the query response time and degree of satisfying of QoS requirement with Entropy Scheduler and Fair Scheduler under different concurrent level of request workload. The results are shown as follows in Fig. 3, Fig. 4 and Fig. 5.

Fig. 3 show that Entropy Scheduler has better performance and a higher degree of satisfying of QoS requirement, which result in improvement of the overall spark analysis server throughput as well (Fig. 4).

However, increasing workload concurrency pose various challenges to the Spark analysis cloud. The cloud experience performance degradation with increasing workload concurrency. There are mainly two reason behind such unstable performance:

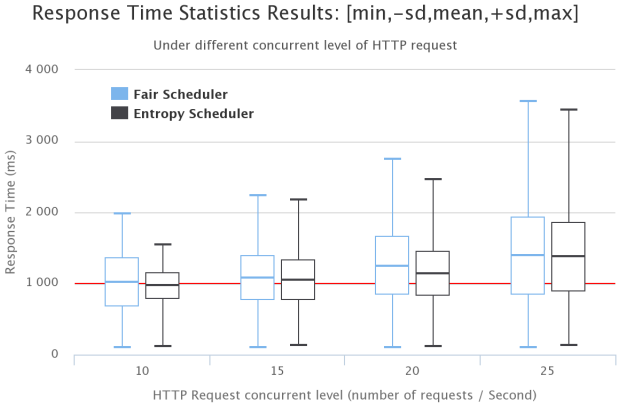

Fig. 3. Experiment 1 : Response time statistics result.

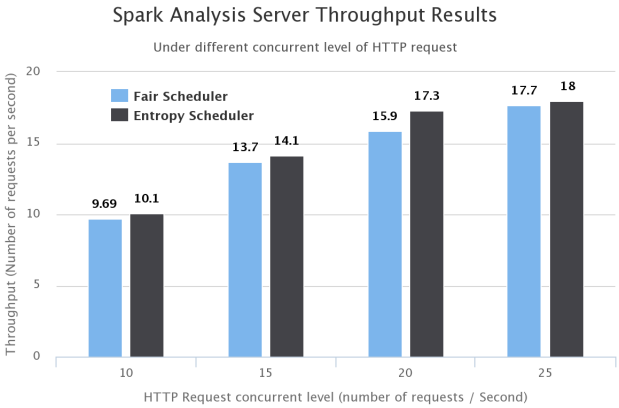

Fig. 4. Experiment 1 : Spark analysis server throughput result.

- First, the loss in cloud performance and stability is due to contention and load interaction among concurrently executing queries [21]. These effects will become worse with more complex workloads.
- Second, the cloud, due to its parallelism and heterogeneity, is a difficult target for achieving low-latency response since poor deployments and/or scheduling lead to performance penalties.

As seen from Fig. 5, although Entropy Scheduler reduce a significant amount of failed requests compared with Fair Scheduler, it still has same performance bottlenecks inhibiting sub-second query response time which motivates future work of other optimization options.

C. Experiment 2 : Load Testing with 100,000 Query Requests at the Concurrent Level of 10

Table II compare the various aspects of load testing result by each scheduler. Our results throughout the Evaluation section show Entropy Scheduler outperform native Fair Scheduler in respect of both efficiency and reliability. On average, in this heterogeneous cluster experiment, Entropy Scheduler shorten the load testing completion time by 23%, reduce the average response time by 23% and standard deviation by 35%, and improve the overall server throughput by 30% compared with native Fair Scheduler.

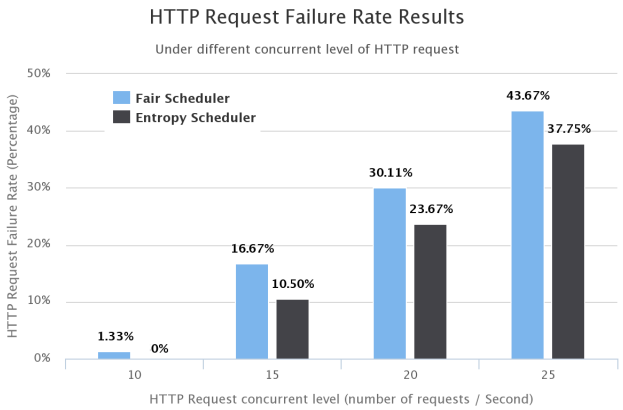

Fig. 5. Experiment 1 : HTTP request failure rate result.

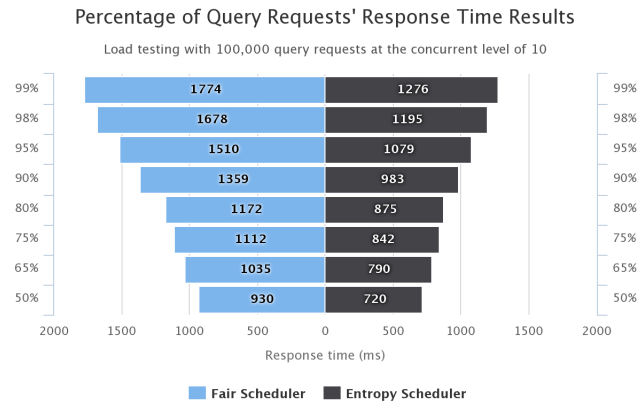

Fig. 6. Experiment 2 : Percentage of the requests served within a certain time (Million Seconds).

TABLE II. EXPERIMENT 2:LOAD TESTING WITH 100,000 QUERY REQUESTS AT THE CONCURRENT LEVEL OF 10

Load Testing Result	Fair Scheduler	Entropy Scheduler
Testing Completion Time (Seconds)	951.52	732.15 (- 23%)
Throughput (Request/Second)	10.51	13.66 (+ 30%)
Number of failed request	75	0
Average Response Time (Million Seconds)	951	732 (- 23%)
Standard Deviation	298.9	194.7 (- 35%)

Fig. 6 indicates that 90% of queries are completed within 1 second under Entropy Scheduler, while only 50% under Fair Scheduler. Such result shows that Entropy Scheduler is more capable for running SAAAS that providing web service with QoS guarantee.

D. Analysis of Results

Our experiments on 3 resources with 18 cores is small-scale, but the experimental results provide intuition for developing new scheduler based on entropy with large-scale of heterogeneous resources. From experiment 1, we have learned the critical bottleneck in current Spark Jobs Scheduling causing by handling high concurrent queries. Compare with native Spark FAIR scheduler, Entropy Scheduler reduces the query Failure Rate by around 7%. The results in Experiment 2 show Entropy Scheduler out-perform FAIR Scheduler for Spark analysis as a service, which will be a starting point for future work, where we hope to run the low-latency query more reliable and efficient.

V. CONCLUSIONS

With the results in the paper, we provide both a concrete solution for a class of emerging systems, as well as a number of ideas valuable for conventional engines running on the cloud.

Based on this, the paper makes the following contributions:

- We show that scheduling low-latency parallel analysis jobs on the Cloud is a nontrivial problem. We identify its performance implications and characterize the elements needed to solve it.
- We introduce the concept of Entropy as a means to characterize and quantify the reliability requirements of the resource.
- We present a novel scheduler that performs three important tasks: (i) Capture the resource CPU utilization and entropy level, (ii) computes the ranking of resource according to both CPU utilization and entropy, and (iii) suggest a scheduling consider the ranking of resource.

Resource allocation in cloud scheduling system is a complex problem, the solution of which requires suitable modeling and complex optimization calculations. The Entropy Scheduler proposed in this paper puts forward an optimization method that is different from the traditional approach. Experiments show that the Entropy Scheduler outperform native Fair Scheduler in respect of both efficiency and reliability. Since the approach of applying Entropy Theory to guide the cloud jobs scheduling is the first attempt in the related literature. Many problems may arise, and many issues remain open. Future work should further examine and expand the entropy method presented in this paper: *a*) We would like to load test the Entropy Scheduler with larger scale of cloud and more complex query workload to ensure it is robust to more complex environment; *b*) We would also like to learn the idea from Omega, Mesos, Sparrow ..., and then transform the Entropy Scheduler from centralized to decentralized to solve the bottleneck problem when meeting with high concurrent query workload; *c*) Also we are currently developing a simulation tools to benchmark all the scheduler supported by Spark, such as YARN and MESOS; *d*) Finally, we plan to research on its application in other similar cloud engines, e.g. MapReduce, Hive, Dremel, Impala.

REFERENCES

[1] Amos, Brandon, and David Tompkins. "Performance Study of Spindle, A Web Analytics Query Engine Implemented in Spark." Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on. IEEE, 2014.

[2] Ousterhout, Kay, et al. "Sparrow: distributed, low latency scheduling." Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, 2013.

[3] Chen, Huankai, Frank Z. Wang, and Na Helian. "A Cost-Efficient and Reliable Resource Allocation Model Based on Cellular Automaton Entropy for Cloud Project Scheduling." International Journal of Advanced Computer Science and Applications 4.4 (2013): 7-14.

[4] Christodoulou, Symeon, Georgios Ellinas, and Pooyan Aslani. "Entropy-based scheduling of resource-constrained construction projects." Automation in Construction 18.7 (2009): 919-928.

[5] Hermenier, Fabien, et al. "Entropy: a consolidation manager for clusters." Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. ACM, 2009.

[6] Chua, Leon O. "Local activity is the origin of complexity." International journal of bifurcation and chaos 15.11 (2005): 3435-3456.

[7] Zaharia, Matei, et al. "Improving MapReduce Performance in Heterogeneous Environments." OSDI. Vol. 8. No. 4. 2008.

[8] Zaharia, Matei, et al. "Spark: cluster computing with working sets." Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. 2010.

[9] Schwarzkopf, Malte, et al. "Omega: flexible, scalable schedulers for large compute clusters." Proceedings of the 8th ACM European Conference on Computer Systems. ACM, 2013.

[10] Hindman, Benjamin, et al. "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." NSDI. Vol. 11. 2011.

[11] Vavilapalli, Vinod Kumar, et al. "Apache hadoop yarn: Yet another resource negotiator." Proceedings of the 4th annual Symposium on Cloud Computing. ACM, 2013.

[12] Matthews, Robert AJ. "The science of Murphy's Law." SCIENTIFIC AMERICAN-AMERICAN EDITION- 276 (1997): 88-91.

[13] Plestys, R., et al. "The measurement of grid QoS parameters." Information Technology Interfaces, 2007. ITI 2007. 29th International Conference on. IEEE, 2007.

[14] Gan, H-S., and A. Wirth. "Comparing deterministic, robust and online scheduling using entropy." International journal of production research 43.10 (2005): 2113-2134.

[15] Botn-Fernndez, Mara, Francisco Prieto Castrillo, and Miguel Vega-Rodrguez. "Nature-inspired algorithms applied to an efficient and self-adaptive resources selection model for grid Practice of Natural Computing (2012): 84-96.

[16] Wolfram, Stephen. "Universality and complexity in cellular automata." Physica D: Nonlinear Phenomena 10.1 (1984): 1-35.

[17] LEON, O. "Local activity is the origin of complexity." International journal of bifurcation and chaos 15.11 (2005): 3435-3456.

[18] Matthews, Robert AJ. "The science of Murphy's Law." SCIENTIFIC AMERICAN-AMERICAN EDITION- 276 (1997): 88-91.

[19] Lambert, F.L.: The Second Law of Thermodynamics. http://www.secondlaw.com, 2005.

[20] Braun, Tracy D., et al. "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems." Journal of Parallel and Distributed computing 61.6 (2001): 810-837.

[21] Gutierrez-Garcia, J. Octavio, and Kwang Mong Sim. "A family of heuristics for agent-based elastic Cloud bag-of-tasks concurrent scheduling." Future Generation Computer Systems (2012).

[22] V. Leis, P. Boncz, A. Kemper, and T. Neumann. Morsel-driven Parallelism: A NUMA-aware Query Evaluation Framework for the Many-core Age. In SIGMOD 14, pages 743754